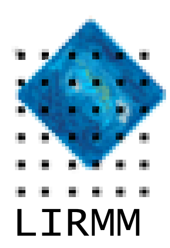
E-NetObject: Un Editeur de Réseaux de Petri à Objets

F. Raclot, David Andreu, Thérèse Libourel Rouge, Robin Passama







E-NetObject : Un éditeur de Réseaux de Petri à Objets

F. Raclot[1], D. Andreu[2], T. Libourel[3], R. Passama[4]

Rapport LIRMM n° 02180

[1] DESS TNI, UMII
[2] LIRMM, département Robotique
[3] LIRMM, département Acquisition et Représentation des Connaissances
[4] LIRMM, département Robotique et Acquisition et Représentation des Connaissances

1. Présentation

1.1. Le Projet Global

Cet éditeur de Réseaux de Petri à Objets s'inscrit dans une chaîne de développement d'applications robotiques.

Ce projet est le fruit d'une réflexion et de travaux déjà amorcés les années précédentes à travers une collaboration entre les départements ROB et ARC du LIRMM. Cette collaboration initiée par David ANDREU avec Thérèse LIBOUREL et Christophe DONY, porte sur la conception d'un système de développement d'applications temps-réel distribuées basées sur les réseaux de Petri à Objets (RdPO).

La partie du projet exposée ici, aborde la conception et la réalisation d'un environnement d'édition de RdPO, modèle à partir duquel sont décrites les applications (cf. [Raclot02]). Ces applications seront ensuite construites à partir de l'assemblage de composants basés sur les RdPO [Passama02]. Cette chaîne est composée des outils suivants :

- Un éditeur de RdPO (Net-Objet) permettant de créer et d'éditer des RdPO.
- Un générateur XML créant un fichier XML décrivant le réseau édité.
- Un compilateur produisant une image C/C++ du RdPO à partir du fichier XML.
- Un joueur, cet élément de la chaîne est celui qui animera le RdPO. Ainsi il devra propager les objets du réseau (Pousse-Jeton), gérer le temps et les actions à l'intérieur des transitions du RdPO, gérer les échanges via le réseau [RR02182].

Voici un schéma descriptif de la chaîne de développement (Figure 1) :

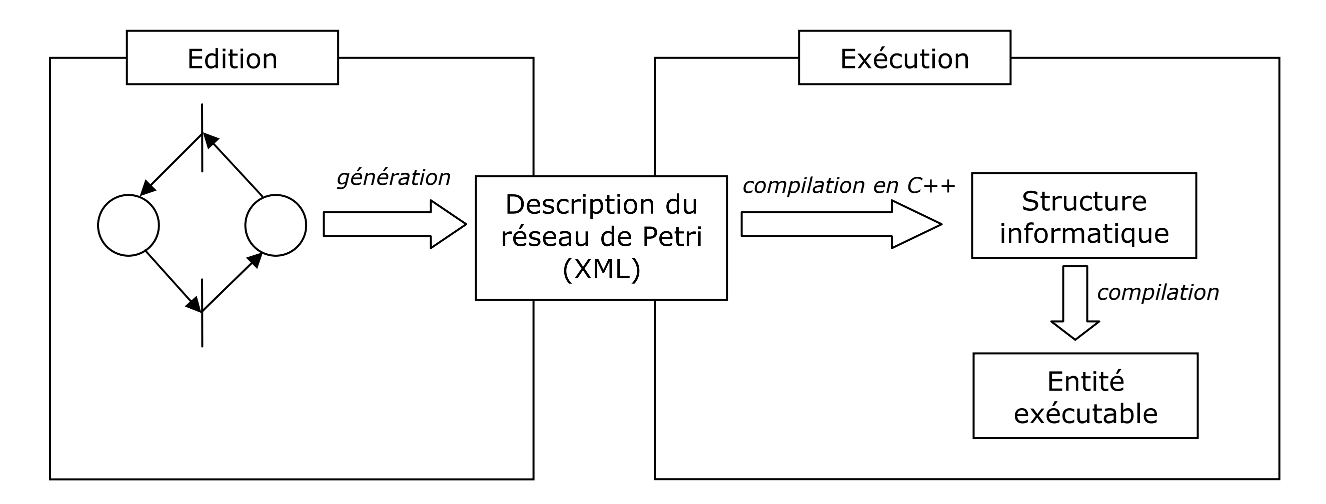


Figure 1 : description de la chaîne de développement

Le manuel d'utilisation de cet éditeur est donné dans [RR02181].

La figure ci-après (cf. Figure 2) décrit le mode de fonctionnement de l'éditeur.

Cette chaîne permet donc à l'utilisateur de saisir le réseau de son choix, de générer l'image XML contenant toute sa description : la structure, le marquage, les classes et l'interface définissant les possibilités de communication avec les autres réseaux. Cette image XML sera déposée à terme sur un nœud matériel via un environnement de déploiement gérant la mise en relation de tous ces composants (cf. [Passama02]).

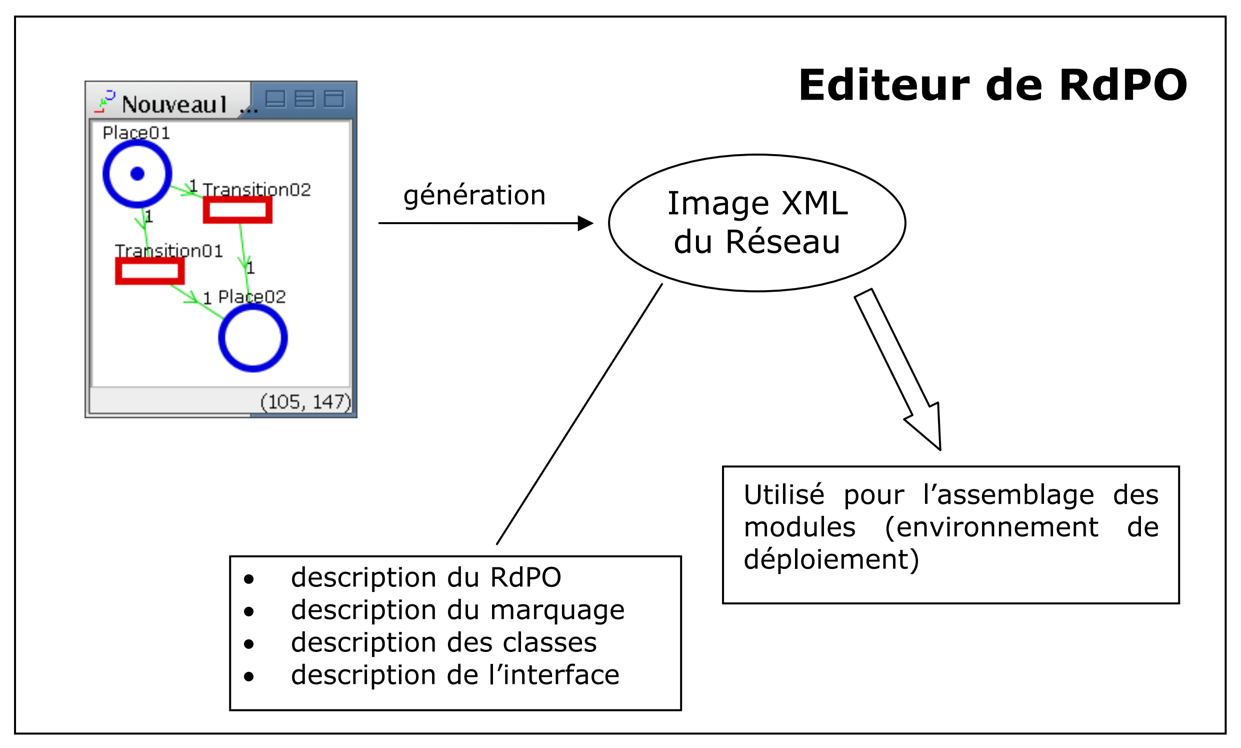


Figure 2 : fonctionnement de l'éditeur

Une fois les images XML des réseaux déployées sur les nœuds matériels, celles-ci seront compilées en langage C/C++. Ceci fait, les sources C/C++ résultant de cette génération seront regroupées avec les sources C/C++ du Joueur de réseau de Petri. Le tout sera compilé en langage machine par un compilateur adapté au système d'exploitation (OS) du nœud matériel cible (cf. Figure 3).

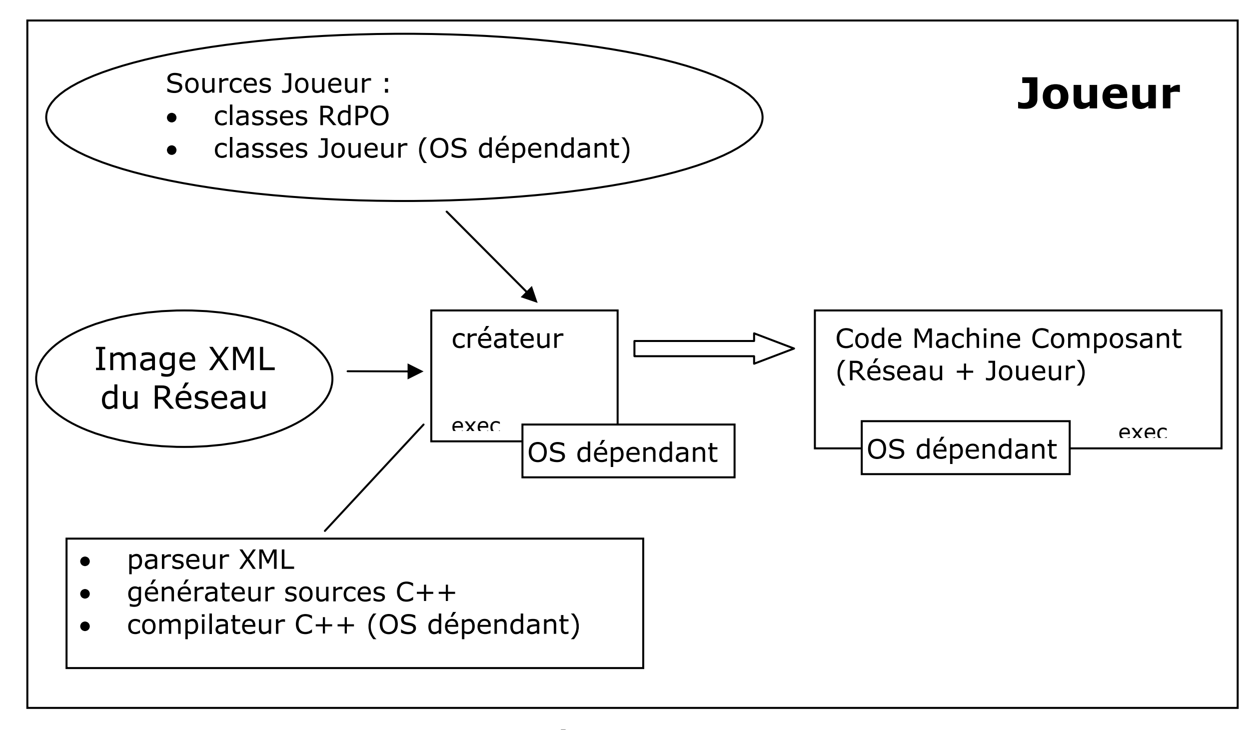


Figure 3 : création d'un composant

2. Contexte

Les réseaux de Petri sont généralement utilisés pour modéliser des systèmes qui sont asynchrones, distribués, non-déterministes et/ou stochastiques. Les Places représentent les états possibles du système, dont l'activité repose sur l'évolution du marquage (distribution des jetons sur les places).

2.1. Les Réseaux de Petri

Nous exposons ici le formalisme des RdPO que nous utilisons dans ce projet. Afin de ne pas faire une description trop lourde du modèle, nous commençons par définir le modèle des Réseaux de Petri simples, ensuite nous expliquerons l'extension qu'apportent les objets au modèle. Enfin, suivra le formalisme classique des RdPO.

2.1.1. Réseaux de Petri simples

Le formalisme des réseaux de Petri est un puissant outil mathématique et sa représentation graphique constitue un bon outil de modélisation [David89]. Un réseau de Petri simple (sans objet) est un graphe composé de Places (les ronds), de Transitions (les rectangles) et d'Arcs orientés (les flèches). Ces derniers relient obligatoirement une Place et une Transition (ou vice-versa), il ne peut donc pas y avoir de nœud (Place ou Transition) liée à un nœud du même type par un Arc (normal ou inhibiteur) (cf. Figure 4).

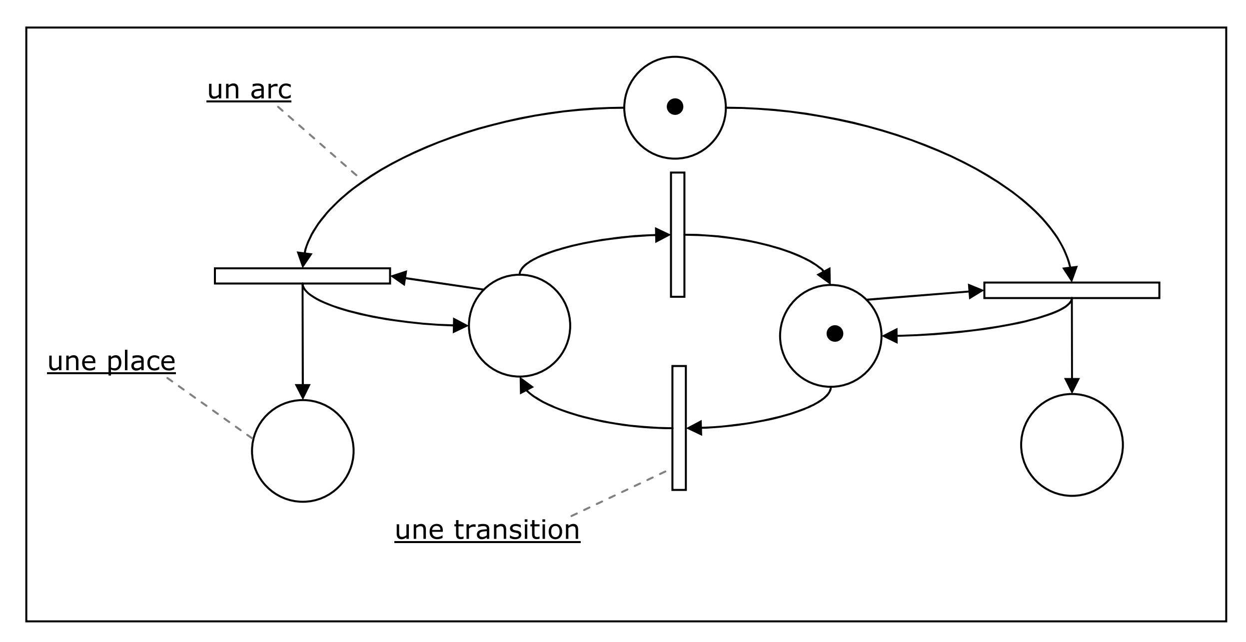


Figure 4 : exemple de RdP simple

Les Places peuvent contenir des Jetons, des poids peuvent être associés aux arcs pour indiquer le nombre de Jetons requis dans la Place amont pour que la Transition soit sensibilisée (cf. Figure 5). Une Transition est dite tirable (franchissable) si elle est suffisamment sensibilisée et si la condition qui lui est associée est vérifiée. Si un Arc inhibiteur est en entrée d'une Transition, celle-ci est tirable si les Places amont des Arcs inhibiteurs ne contiennent pas de Jeton. Ces Arcs spéciaux ne peuvent pas exister en sortie d'une Transition, de plus leur présence seul en entrée d'une Transition n'a pas de sens.

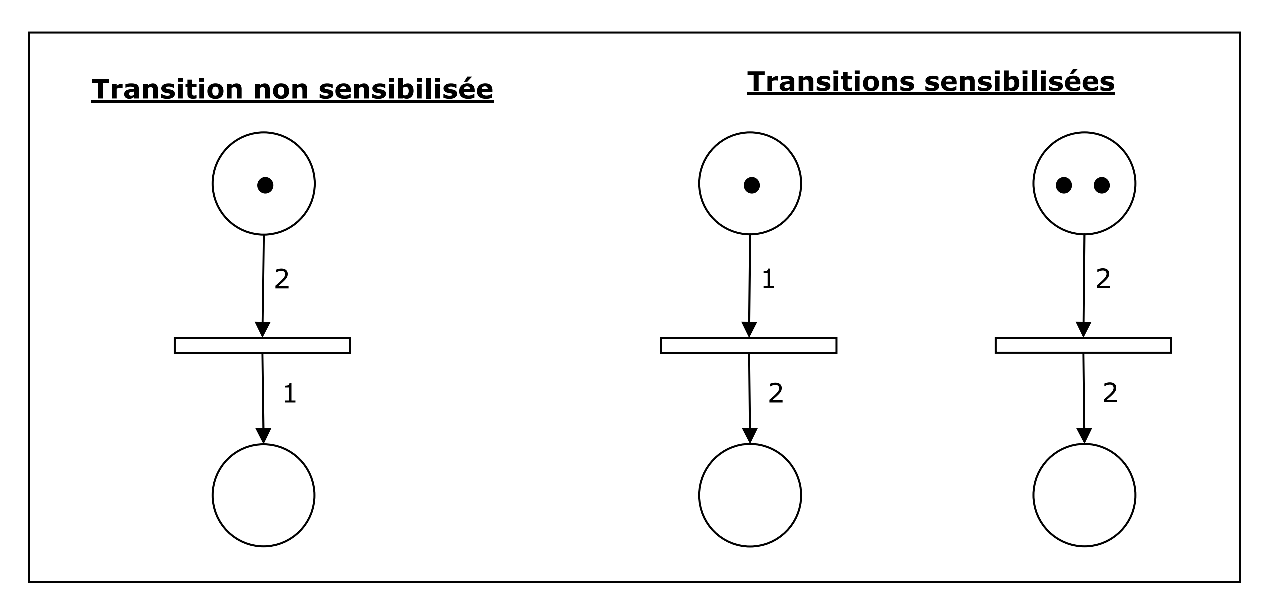


Figure 5 : exemples de sensibilisations

Le franchissement de la transition induit le retrait (respectivement l'ajout), de Jetons sont dans les Places amont (respectivement aval) en fonction de la pondération des Arc les reliant à la transition.

2.1.2. Ajout des objets

Le modèle de réseaux de Petri précédent présente une insuffisance au niveau de son pouvoir d'expression. En effet dans ce modèle les Jetons ne contiennent pas de données et ne sont pas différentiables (jetons dits banalisés).

Pour palier à cette limitation, la notion de tuple d'objets est venue en remplacement du jeton. Des classes d'objets sont associées au réseau, éventuellement organisées en hiérarchie, il est alors possible de contrôler le type des objets qui participent au marquage d'une Place, de la même manière les Arcs peuvent filtrer les types des objets qui pourront éventuellement franchir une Transition. Une étiquette associée à l'arc permet de spécifier le type de jeton (type d'objets) admis par cet arc. Ce type de jeton doit bien sur être admis par le place amont : pour chaque place l'ensemble des objets (jetons) admis est spécifiée. Au niveau des transitions les conditions ne peuvent porter que sur les jetons entrants. Les actions peuvent, elles, porter sur les jetons entrants (provenant des places amont) et sortants (destinés aux places aval). Les transitions supportent les conditions (ou garde) d'évolution du graphe (expression booléenne) ainsi que les actions à générer lors du franchissement.

A l'heure actuelle les RdPO ne sont quasiment utilisés qu'en modélisation, en effet le passage à l'implémentation se fait la plupart du temps manuellement. Ceci est très coûteux en temps, et peut aussi être générateur d'erreurs.

2.1.3. Le modèle RdPO [Valette00]

Un réseau de Petri est constitué de :

- Un ensemble fini *P* de Places.
- Un ensemble fini *T* de Transitions.
- Un ensemble fini C_{lass} de Classes, éventuellement organisé en une hiérarchie et définissant pour chaque Classe un ensemble d'attributs.
- Un ensemble de variables *V* typées par C_{lass}.
- Une fonction *place précédente* appelée *Pre* qui à chaque Arc d'entrée d'une Transition fait correspondre une somme formelle de n-uplets d'éléments de *V*.
- Une fonction *place suivante* appelée *Post* qui à chaque Arc de sortie d'une Transition fait correspondre une somme formelle de n-uplets d'éléments de *V*.
- Une application A_{tc} qui à chaque Transition associe un ensemble de conditions qui font intervenir les variables formelles associées aux arcs d'entrée et les attributs des classes correspondantes.

- Une application A_{ta} qui à chaque Transition associe un ensemble d'actions qui font intervenir les variables formelles associées aux arcs d'entrée et de sortie, les attributs et les méthodes des classes correspondantes.
- Une application M_0 est le marquage initial qui associe à chaque Place une somme formelle de n-uplets d'instances d'objets appelés Jetons (les objets doivent être représentés par des identificateurs, leur nom par exemple).

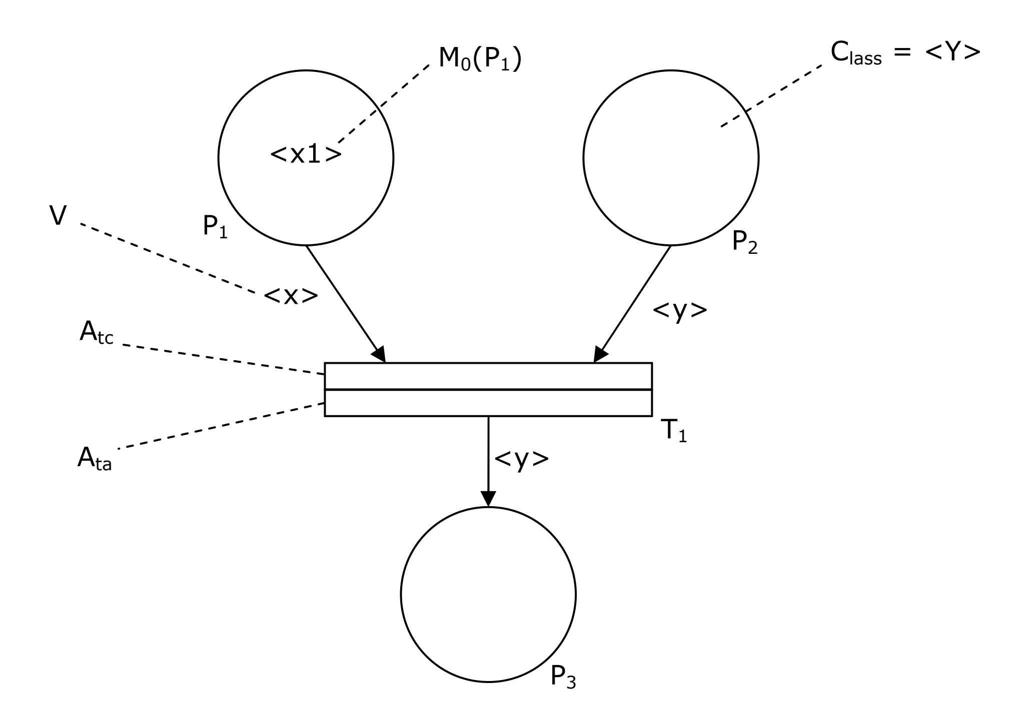


Figure 6

2.1.4. Le modèle RdPO de l'éditeur

Nous allons présenter les particularités de notre modèle RdPO (modèle édité), par rapport au RdPO « général » que l'on vient de définir.

a) *Particularités*

1) La Transition

Les transitions T sont composées de plusieurs parties, autrement appelées "blocs".

- Le premier bloc est celui relatif aux prédicats : il supporte l'expression de la conditions de franchissement.
- Le deuxième bloc est dédié à l'aspect temporel : il permet d'exprimer des intervalles de franchissement d'une transition. Un intervalle temporel I est constitué de deux bornes [a, b] : a et b étant respectivement la date de tir au plus tôt et la date de tir au plus tard.
- Le troisième bloc est dédié au verrouillage de la transition. Par le biais de variables globales booléennes appelées "verrous", il est possible d'interdire le franchissement d'une transition ou d'un ensemble de transitions.
- Le quatrième bloc est relatif aux actions. Les actions sont en effet associées aux transitions et exécutées lors du tir de la transition. Une action à durée non nulle (i.e. non considérable comme "instantanée") induit un délai de franchissement de la transition… cela nous ramène à un modèle T-temporisé. Le jeton ne sera déposé dans la place aval uniquement lorsque l'action sera terminée. Cette représentation, proche d'un modèle T-temporisé, est assimilable à un modèle au sein duquel une transition peut être affinée (raffinement de transition) par une séquence transition-place-transition.

Nous proposons la possibilité de spécifier l'exécution des actions associées à une transition sous forme de threads (i.e. exécution en parallèle de l'exécution du graphe). Notons que la transition concernée est assimilée à une transition temporisée.

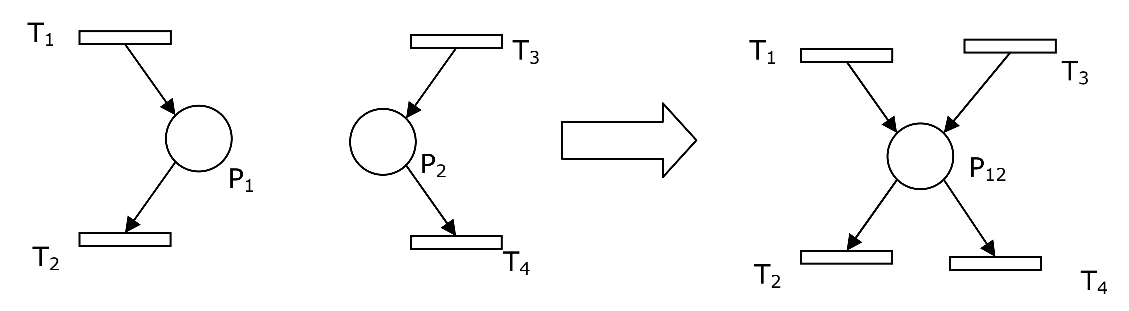
Par ailleurs, il est possible d'attribuer des niveaux de priorité aux transitions. Ces niveaux (2^{32}) de priorité ne sont pris en compte que dans le cadre de la résolution de conflit (cf. [RR02182]).

2) La Structuration

- Les sous réseaux ont pour but de faciliter la décomposition et de fait la réutilisabilité d'un réseau de Petri. Il est ainsi possible de les composer afin de donner naissance à un modèle composite. Les moyens de composition utilisent les arcs de fusion et/ou les places de communication (cf. ci-dessous).

- Les arcs de fusion permettent, comme leur nom l'indique, de fusionner deux places (de communications, i.e. d'entrée ou de sortie) ou deux transitions entre elles.

Fusion des Places



Fusion des Transitions

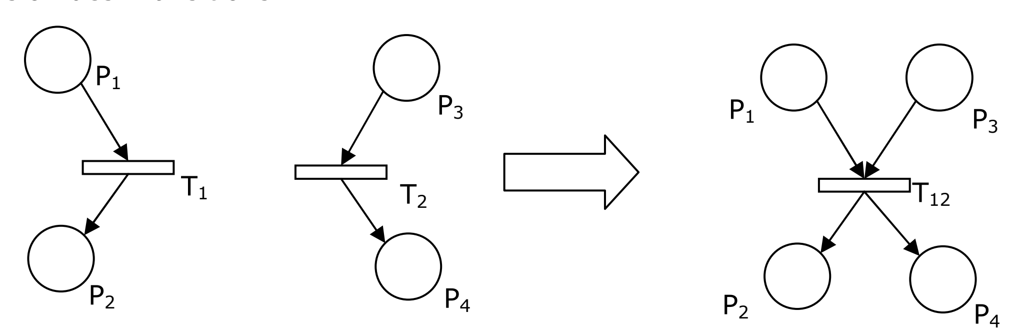


Figure 7 : Fusion des Nœuds d'un réseau

- La communication. Il existe trois types de places P : Interne, Entrée, Sortie. Cette distinction permet d'exprimer explicitement les places d'interactions entre modèles qu'ils soient exécutés sur un même joueur (au sens processus) ou sur des joueurs différents (sur un même processeur ou non) connectés par un réseau de communication. Une place est par défaut une place interne ; il s'agit d'un état interne du modèle. Une place d'entrée est une place de réception ; il s'agit d'une place dans laquelle est reçu un message. Une place de sortie est une place d'émission ; il s'agit d'une place dans laquelle est déposé un message à émettre. Ces deux derniers types de places ont la possibilité d'être connecté entre eux via des arcs de fusion (cf. ci-dessus).

3) Les Marques du Réseau et les Variables Globales

Il est possible d'éditer deux types de modèles, voire de les coupler. Ces modèles sont les réseaux de Petri pondérés et les réseaux de Petri à Objets ; tous deux pouvant être temporels.

Pour les réseaux de Petri à Objets, les marques sont des objets dont les classes sont définies par l'utilisateur. La hiérarchisation des classes (et donc l'héritage, le polymorphisme, etc.) est possible. Sur ce modèle les arcs permettent de désigner, par le biais de variables, les types d'objets à faire transiter à travers la transition. A chaque place

est associé un ensemble des classes admises définissant les classes d'objets que peut contenir la place.

Pour les réseaux de Petri pondérés les marques sont des jetons banalisés. Une classe spécifique prédéfinie (BANALISE) permet l'utilisation de jetons banalisés. Dans ce cas, les arcs reliant les places et les transitions supportent l'expression de la pondération, i.e. du poids associé.

Une place ne peut contenir que l'un ou l'autre des types, i.e. soit des jetons banalisés, soit des jetons objets.

Au-delà des jetons objets ou des jetons banalisés manipulés au sein du modèle, certaines variables et/ou méthodes globales sont accessibles via toutes les transitions du graphe. A titre d'exemple considérons les variables verrous associées aux transitions. Une variable verrou V_i peut verrouiller n transitions, si elle a été associée à ces n transitions. Pour cela, une transition peut appeler une méthode globale qui teste ou modifie la valeur du verrou considéré.

Toutes ces variables et/ou méthodes sont associées à une classe globale, nommée GLOBAL_CLASS ; l'utilisateur pourra enrichir cette classe (tant sur les méthodes que sur les attributs).

b) *Les éléments*

1) Les Classes

Les Classes d'objets définies dans ce modèle sont organisées en une hiérarchie de spécialisation simple, toute Classe ne pouvant être fille que d'une seule Classe mère. Une Classe est composée d'un ensemble fini d'attributs ainsi que d'un ensemble fini de méthodes. Les attributs et les méthodes de ces classes ne sont définis que sur les types primitifs (entier, réel, booléen, chaîne de caractère et énumérations).

Dans la hiérarchie pré-existent deux classes, BANALISE et GLOBAL_CLASS :

- BANALISE sert uniquement à permettre l'utilisation comme Jeton d'instance de cette Classe : les Jetons correspondant sont alors dits « banalisés » c'est-à-dire non tuples d'Objets, c'est-à-dire ceux utilisés dans les RdP simples.
- GLOBAL_CLASS sert uniquement à la définition de fonctions et de variables globales que l'utilisateur décide d'utiliser pour la globalité de son réseau.

2) Les Jetons et les Objets

Tout d'abord les Objets, ce sont eux qui représentent les instances des Classes définies précédemment.

Le Jetons eux, sont représentés par un multi-ensemble non ordonné fini d'Objets.

3) Place

Les Places sont des entités nommées du réseau et qui peuvent recevoir des Jetons (au sens large c'est-à-dire des tuples d'Objets). Le type des Objets autorisés dans les Jetons potentiels de la Place est indiqué dans celle-ci. Les Jetons effectifs constitueront le marquage de la Place. Le marquage général du réseau sera l'ensemble des marquages de ses Places.

Toute Place est reliée à une ou plusieurs Transitions par des Arcs orientés. Selon le nombre d'Arcs amont/aval la Place a un type : interne (n Arcs aval/amont n>0), isolée (0 Arcs aval/amont), source (0 Arcs aval, n Arcs amont n>0) ou puits (n Arcs aval n>0, 0 Arcs amont).

En plus de ces Places, il existe celles d'entrées et celles de sorties. Celles-ci permettent d'interfacer le réseau avec d'autres réseaux.

4) Arc

Un Arc relie une Place et une Transition ou vice-versa, dans le premier cas c'est un Arc Pre, dans l'autre un Arc Post. Cette entité porte plusieurs Etiquettes qui sont des multi-ensembles non ordonnés de nom de variables associées à des Classes, cette liste définit les Jetons qui peuvent être candidats au franchissement d'une Transition dans le cas d'un Arc Pre ; cette Etiquette définit les Jetons qui seront déposés dans une Place dans le cas d'Arcs Post.

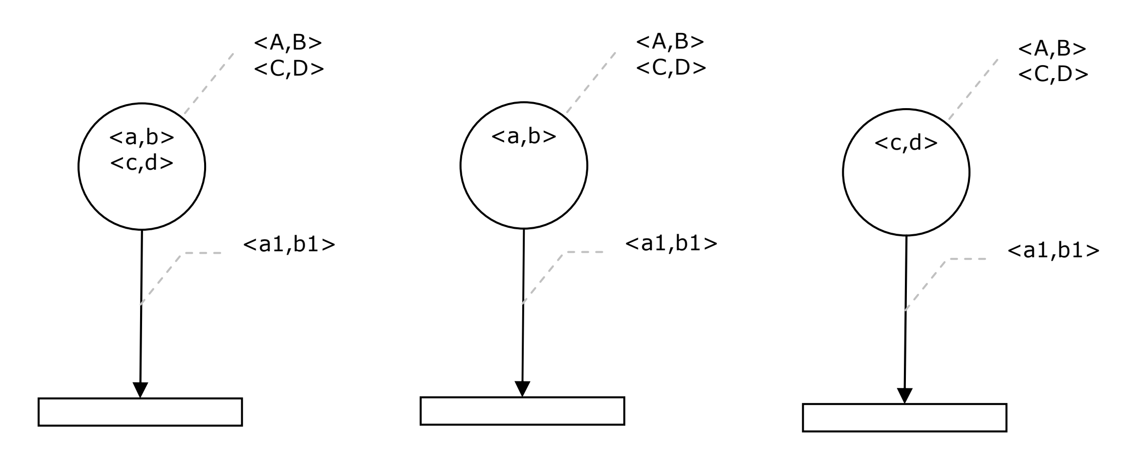
On distingue trois types d'Arcs, les arcs « classiques » (ou simples), les Arcs de test et les Arcs inhibiteurs.

5) Arc de Fusion

Un Arc de Fusion a pour but de permettre la fusion entre deux nœuds du réseau. Les règles définissant cette fusion sont données ci-après.

6) Transition

Cette entité a pour but, en conjonction avec les Arcs, de « filtrer » et d'agir sur les Jetons qui peuvent la traverser. Pour cela, elle est munie d'un ensemble (éventuellement vide) de conditions (appelé aussi garde), d'un ensemble (éventuellement vide) d'actions et éventuellement d'un verrou et/ou d'une fenêtre temporelle. Une Transition est sensibilisée si l'un des ensembles d'Objets représenté par un Jeton présent dans une Place amont est égal à l'un des ensembles de Classes représenté par une Etiquette portée par l'Arc reliant cette Place et la Transition. Voici un exemple (cf. Figure 8) :



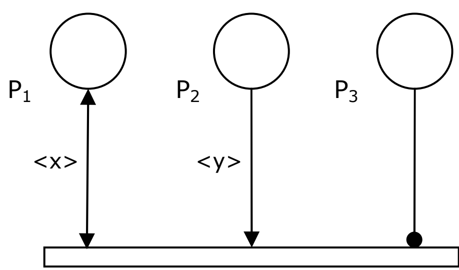
Transitions sensibilisées *Transition non sensibilisée*

<A,B> : *Classes acceptées par la Place*
<a,b> : *Jetons se trouvant dans la Place*
<a1,b1> : *Etiquette de l'Arc*

Figure 8 : exemples de sensibilisations par des jetons « objets »

En généralisant à une transition comprenant les différents types d'Arcs en entrée (« classique », test et inhibiteur), la transition est sensibilisée si :

- P_1 contient n jetons <x>
- P_2 contient n jetons <y>
- P_3 ne contient aucun jeton



Remarque : le jeton <x>, amené par l'Arc de test reliant la transition à la place P1, ne participe pas vraiment au tir. Seule sa présence (respectivement son absence) permet (respectivement empêche) le tir et le jeton n'est pas affecté par ce tir.

La condition de la Transition est un ensemble de prédicats portants sur les variables des Arcs entrants. Si les Objets contenus dans le Jeton passent les tests avec succès le Jeton pourra aller dans la Place aval (cela dépend du ou des Arcs sortants de la Transition).

L'action d'une Transition est un ensemble d'expressions concernant les variables et/ou méthodes des Arcs entrants ou sortants, cela peut être aussi l'appel à une fonction de GLOBAL_CLASS. Il est ainsi possible de manipuler les Objets contenus dans un Jeton.

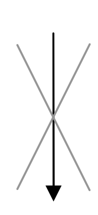
Lorsque les Objets d'un Jeton ont passé les tests ceux-ci pourront être ré-utilisés afin d'en construire un nouveau, d'autres pourront être créés, tout dépend des Etiquettes portées par les Arc avals.

c) *Les Règles*

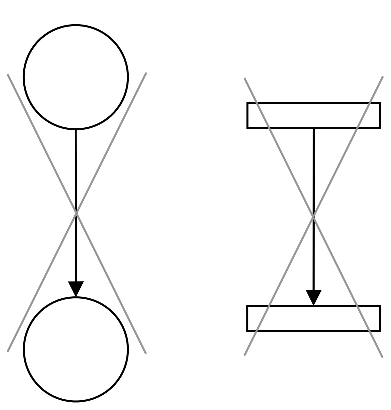
1) Relatives à la Structure

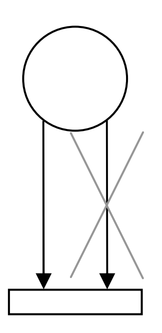
- Règles structurelles de base :

ERREURS

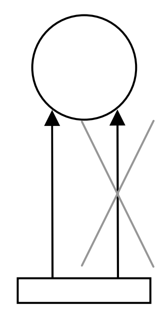


Les nœuds sont obligatoirement reliés par des arcs ; les arcs n'existent que lorsqu'ils connectent deux nœuds en respectant les contraintes ci-après. Un arc isolé, i.e. sans nœud amont ni aval est impossible.

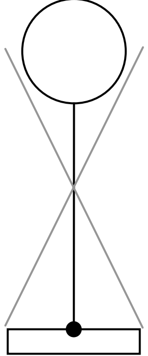


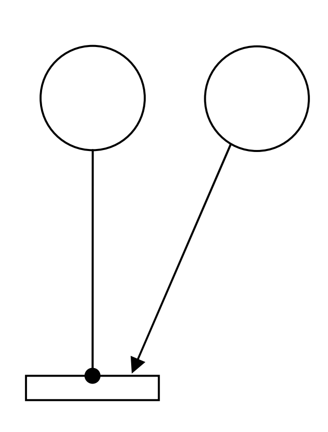


Il y a obligatoirement alternance dans la connexion des nœuds, i.e. il est interdit de relier deux transitions entre elles ou deux places entre elles (seul cas d'exception : les arcs de fusion).

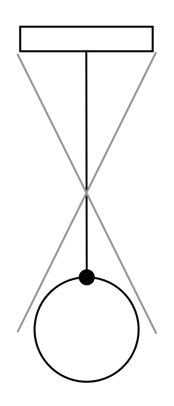


Deux nœuds (une place et une transition) peuvent être reliés dans les deux sens mais un seul et unique arc dans chaque sens.

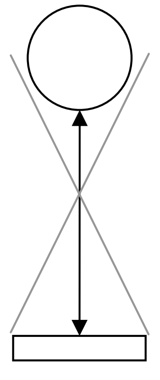


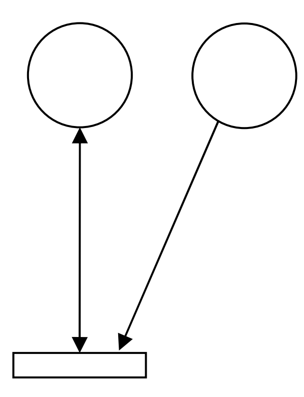


Un arc inhibiteur est obligatoirement accompagné d'un arc simple en entrée d'une transition, en effet, sinon cela n'aurait pas de sens.



Un arc inhibiteur ne peut être en sortie d'une transition.

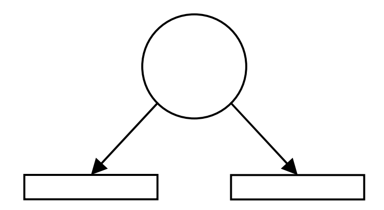




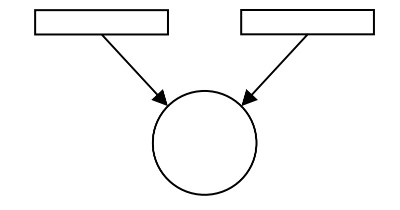
Un arc test est obligatoirement accompagné d'un arc simple en entrée d'une transition, en effet, sinon cela n'aurait pas de sens.

Figure 9

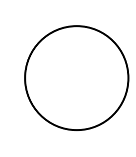
AVERTISSEMENTS



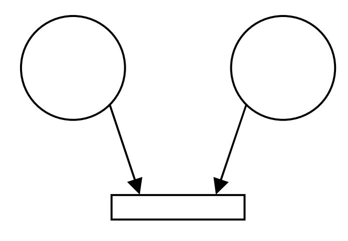
Place interne "source" (place sans arc entrant).



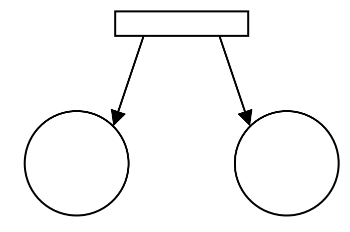
Place interne "puit" (place sans arc sortant).



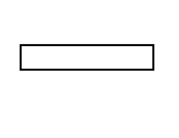
Place isolée (sans arc entrant ni sortant).



Transition "puit" (transition sans arc sortant).



Transition "source" (transition sans arc entrant).



Transition isolée (sans arc entrant ni sortant).

Figure 10

- Règles de la fusion des places :

La fusion des places n'est possible qu'entre places de communication (i.e. d'entrée ou de sortie) ou entre places internes, il n'est donc pas possible de fusionner une place de communication avec une place interne. Deux places de communication (ou interne) peuvent être fusionnées si elles acceptent le même type de Jetons. La fusion de deux places de sorties résulte en une place de sortie, la fusion de deux d'entrées en une place d'entrée. Il est aussi possible de fusionner une place d'entrée et une de sortie. La résultante est une place interne mutante, c'est-à-dire qu'elle reste fusionnable avec d'autres places de sortie, d'entrée ou interne mutante. La fusion de deux places internes résulte en une place interne. Dans tous les cas, les types de jetons acceptés par la place restent les mêmes. Voici les caractéristiques du résultat de la fusion de deux places :

- L'ensemble des transitions amonts (respectivement avals) est l'union des ensembles des transitions amonts (respectivement aval) de chaque place.
- Le marquage initial est la somme des marquages initiaux des places concernées.

- Règles de la fusion des transitions :

En plus de la fusion des places, il est possible de fusionner les transitions. La fusion des transitions impose moins de contraintes que celle des places. L'ensemble des variables portées par les étiquettes des arcs doivent avoir une intersection nulle. Si l'une ou les deux transitions sont temporisées, l'intersection des intervalles de temps doit être non nulle (l'intervalle de temps d'une transition non temporisée est considéré comme étant [0, +∞ [). Soit T_f la fusion de T_1 et T_2 :

- L'ensemble des places amonts (respectivement aval) de T_f est l'union des ensembles de places amont (respectivement aval) de T_1 et T_2.
- La condition portée par T_f est le ET des conditions des transitions initiales : $A_{tc}(T_f) = A_{tc}(T1) \wedge A_{tc}(T_2)$.
- L'action portée par T_f est le ET des actions portées par les transitions initiales : $A_{ta}(T_f) = \{A_{ta}(T_1) ; A_{ta}(T_2) ;\}$.

- L'intervalle de temps porté par T_f est l'intersection des intervalles des transitions initiales : $T_{emps}(T_f) = T_{emps}(T_1) \cap T_{emps}(T_2)$. Si l'intersection est nulle, une erreur est signalée et aucun intervalle n'est associé à Tf.
- La priorité de T_f est la plus haute des priorités des transitions initiales : $P_{riorite}(T_f) = Max(P_{riorite}(T_1), P_{riorite}(T_2))$.
- Le verrou de T_f est un OU entre les verrous des transitions initiales : $V_{errou}(T_f) = V_{errou}(T_1) \vee V_{errou}(T_2)$.

2) Relatives à l'Interprétation

Il existe également des règles de construction garantissant la cohérence de l'interprétation du graphe :

- Sur les transitions : L'ensemble des conditions porté par une transition est traité comme une conjonction de conditions lors de l'exécution du modèle. En effet chaque condition de l'ensemble est lié avec les autres par un ET logique. Les seuls attributs et méthodes utilisables dans la condition sont ceux des Jetons entrants ainsi que ceux de GLOBAL_CLASS. Les attributs et méthodes utilisables dans les actions sont ceux des Jetons entrants et sortants ainsi que celles de GLOBAL_CLASS.

- Sur les arcs : L'intersection des ensembles de variables d'entrée (resp. de sortie) portés par chaque arc d'entrée (resp. de sortie) d'une même transition doit être vide. C'est à dire qu'une transition ne prend pas en entrée (resp. ne produit pas en sortie) 2 variables identiques provenant de 2 arcs distincts. Les seules étiquettes pour un arc pré sont celles des classes admises dans la place amont et pour un arc post ce sont celles des classes admises dans la place aval.

3. Conception

3.1. Editeur de RdPO

L'éditeur doit permettre à l'utilisateur de créer et d'éditer des RdPO. Il doit posséder toutes les caractéristiques d'un éditeur classique comme :

- couper/copier/coller
- annuler/rétablir
- les fonctions de sauvegarde, d'ouverture etc.

En tant qu'éditeur de réseaux de Petri il doit pouvoir :

- donner la possibilité de créer tous les éléments structurels du modèle y compris les Arcs inhibiteurs (et les MacroPlaces dans le futur).
- respecter les contraintes liées à la structure des réseaux de Petri.
- donner la possibilité de créer des sous réseaux à partir de réseaux déjà sauvegardés afin que l'utilisateur puisse les réutiliser.
- créer des modèles génériques. Cette fonctionnalité permet de protéger l'édition de modèles qui seront amenés à être instanciés.
- générer un réseau sous une forme textuelle (XML dans le cas présent) afin que celui-ci puisse être converti facilement et exporté vers des logiciels tiers.

Etant donné que cet éditeur doit être évolutif, il était plus cohérent de le développer dans un langage orienté objet. Pour des raisons de portabilité, et parce que la performance n'était pas une donnée fondamentale du cahier des charges de l'éditeur, le langage de programmation Java a été préféré au C++. De plus il est à noter que la structure et l'interface ont été conçues de manière indépendante, ainsi il est possible d'avoir une interface totalement différente reprenant la structure.

3.1.1. Les entités d'un RdPO

Voici le diagramme des classes des entités contenues dans un RdPO :

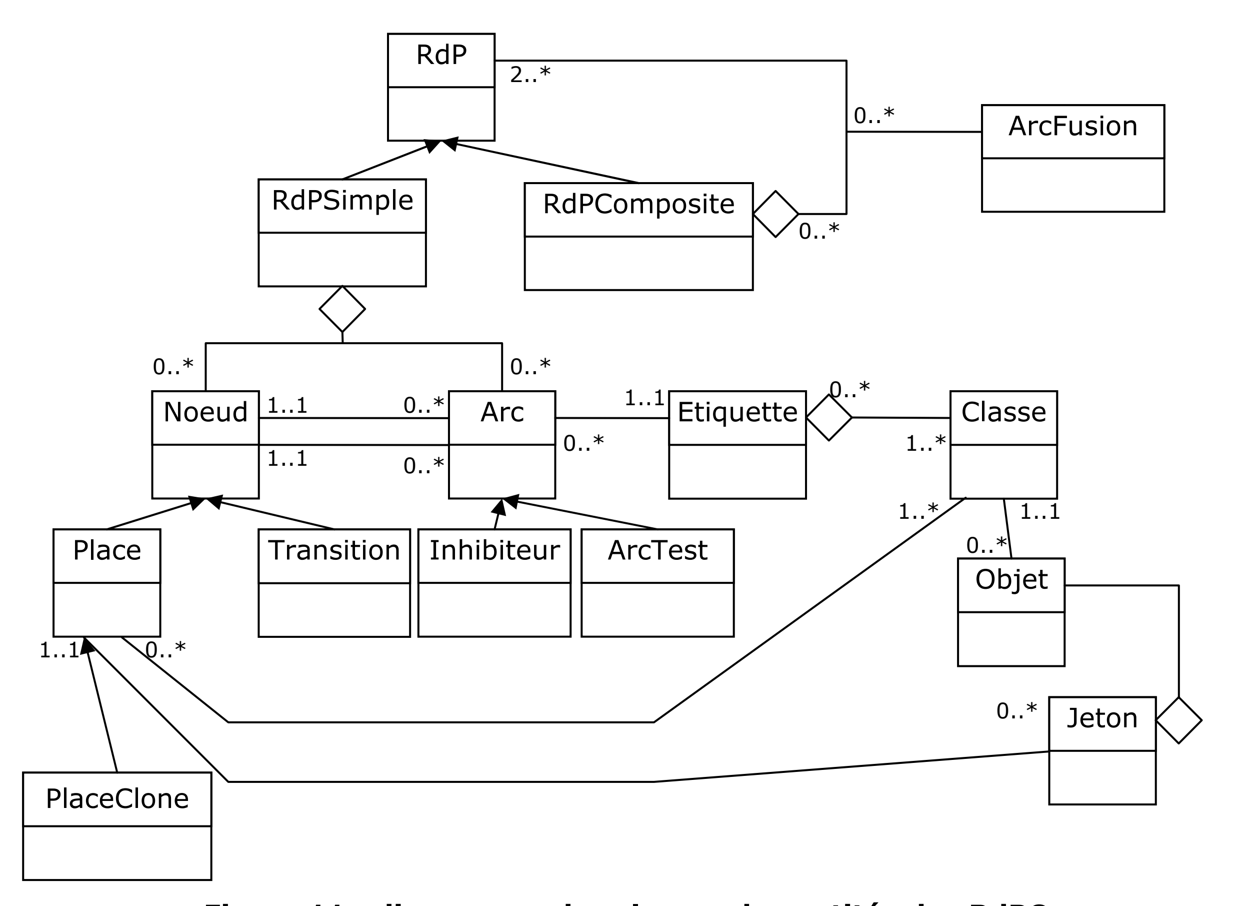


Figure 11 : diagramme des classes des entités des RdPO

Les éléments impliqués dans ce diagramme sont :

a) *Arc*

Un Arc relie deux Noeuds entre eux, mais les contraintes inhérentes au modèle des réseaux de Petri sont vérifiées dans l'interface. Un Arc connaît son Noeud de départ et son Noeud d'arrivée ainsi que la liste d'étiquettes qui lui sont associées.

b) *ArcTest*

Cette classe implémente les Arcs de Test, c'est l'une des sous classes de Arc.

c) *Inhibiteur*

Cette classe est l'autre sous-classe de Arc, elle caractérise les Arcs inhibiteurs.

d) *ArcFusion*

Ces Arcs servent à fusionner deux Places ou deux transitions de deux réseaux différents. Les Places doivent avoir les mêmes caractéristiques, à savoir accepter les mêmes types de Jetons.

e) *Classe*

Cette classe implémente les Classes du réseau. Les instances de ces Classes sont les Objets que nous retrouvons dans les Jetons. Elle comporte une liste d'attributs ainsi qu'une liste de méthodes, elle connaît aussi sa Classe mère.

f) *Etiquette*

Cette entité représente les Jetons que peut accepter une Place, ou qui peuvent passer par un Arc.

g) *Noeud*

Cette classe réunit les caractéristiques des Nœuds du réseau, à savoir les Places et les Transitions. Elle possède une liste des Arcs entrants et une des Arcs sortants.

h) *Objet*

Ils représentent l'instanciation des Classes du réseau.

i) *Jeton*

Un Jeton est un tuple d'Objets, il est utilisé pour le marquage des Places.

j) *Place*

Une Place est reliée à une ou plusieurs Transitions via des Arcs. La Place est l'entité qui contient les Jetons, qui ne peuvent y être placés que si la Place les admet. C'est une sous-classe de Nœud.

k) *PlaceClone*

Ce type de Place a pour but de clarifier un réseau où les Arcs sont nombreux. Pour cela l'utilisateur a la possibilité de cloner une Place, le clone a les mêmes caractéristiques que la Place d'origine sauf que, visuellement, la Place et son clone n'ont pas les mêmes Arcs entrants et sortants.

l) *Transition*

Une Transition est reliée à une ou plusieurs Places via des Arcs. La Transition est l'entité qui conditionne le passage des Jetons (tir de la Transition) et qui peut agir sur ceux-ci. C'est une sous-classe de Nœud.

m) *RdP*

RdP est l'entité abstraite représentant un réseau de Petri à objets. Un RdP peut être un modèle générique, une fois exporté en tant que tel, il ne pourra être instancié qu'en étant sauvegardé après avoir été importé en tant que RdP normal. Si l'utilisateur veut modifier le RdP générique, il devra l'importer à partir du disque dur puis l'exporter de nouveau en tant que modèle générique. Les modifications apportées à un modèle générique ne sont pas répercutées dans les instances crées auparavant.

n) *RdPSimple*

Un RdPSimple est l'entité qui contient les Nœuds, les Arcs, les Classes d'un réseau de Petri.

o) *RdPComposite*

Un RdPComposite est un réseau composé de plusieurs réseaux, ou alors un sous-réseau (il est à noter qu'il est impossible de sauvegarder un sous-réseau). Il contient au moins un RdPSimple.

3.1.2. L'interface de l'éditeur

Concernant le design de l'interface, deux possibilités s'offraient à nous. Une première, classique, avec une fenêtre principale contenant toutes fenêtres des réseaux, une autre avec un ensemble de fenêtres indépendantes. Cette dernière possibilité a été retenue car la gestion des sous-fenêtres est faite directement par Java alors, que les fenêtres principales sont gérées par le système.

Voici un schéma de l'interface :

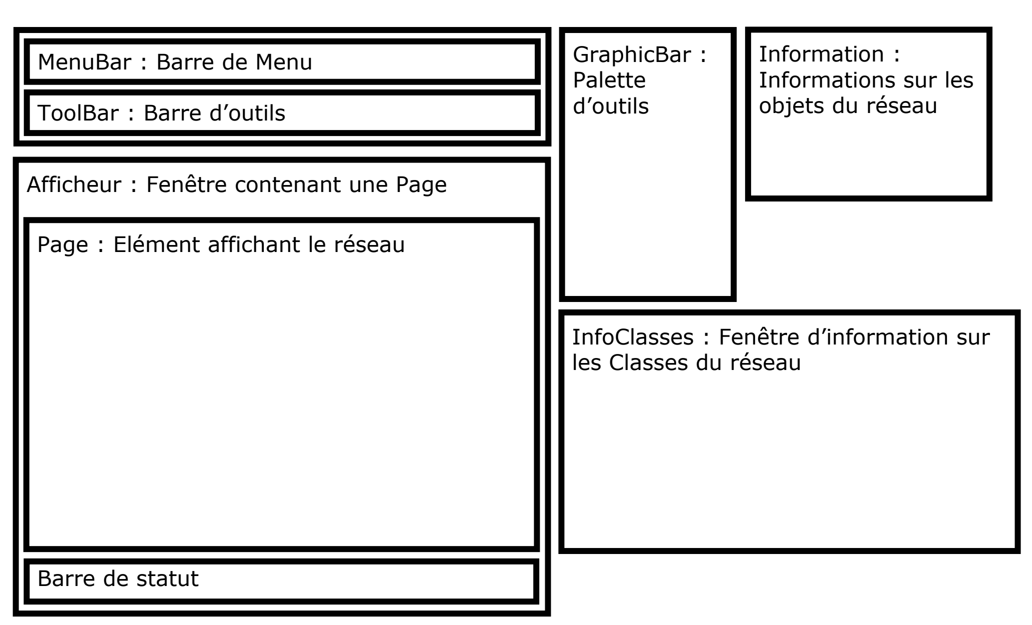


Figure 12 : schéma de l'interface choisie

Liste des fenêtres :

a) *Editeur*

Cette fenêtre est la principale de l'interface elle centralise la plupart des informations et gère la liste des réseaux ouverts. Elle est composée d'une barre de menu, et d'une barre d'outils.

1) MenuBar

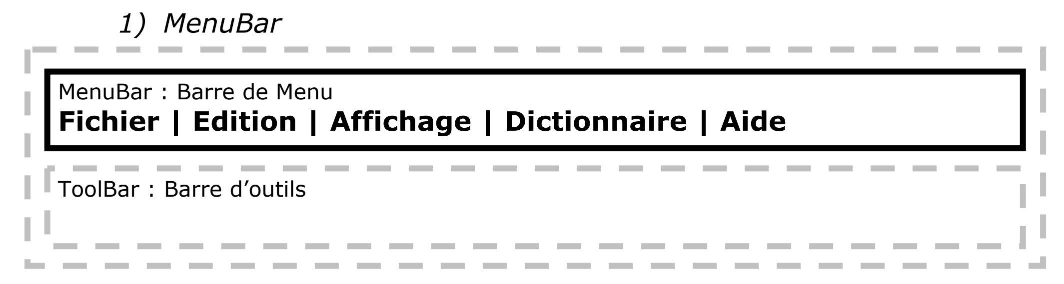


Figure 13 : schéma de la barre de menus

Cette barre de menu comporte des menus classiques comme le menu Fichier, Edition, Affichage, Aide et d'autres menus comme Dictionnaire, en voici une description :

- Fichier : Ce menu possède les actions habituelles comme Nouveau, Ouvrir, Enregistrer, Imprimer. Ce menu a en plus des actions spécifiques aux réseaux de Petri, comme Importer/Exporter un modèle générique, Générer une image XML du réseau, Compiler un réseau en C++ (cette fonctionnalité est appellée à disparaître dans le futur) et gérer les librairies de Classes.
- Edition : Cette partie permet de Couper/Copier/Coller/Supprimer un objet sélectionné. Elle permet aussi d'Annuler où de Rétablir une action.
- Affichage : Ce menu permet d'afficher ou de cacher les fenêtres de l'interface comme la GraphicBar (Palette d'outils), la fenêtre d'information générale, ou celle spécifique aux classes. Dans un réseau, il permet aussi d'afficher directement des informations sur la Page comme le noms des objets, le marquage sur les places ainsi que les images miniatures des sous-réseaux.
- Dictionnaire : Ce menu donne la possibilité de modifier et de créer des éléments du réseau.
- Aide : Classiquement, ce menu donne accès à l'aide du logiciel au format HTML ainsi qu'a des informations le concernant.

2) ToolBar

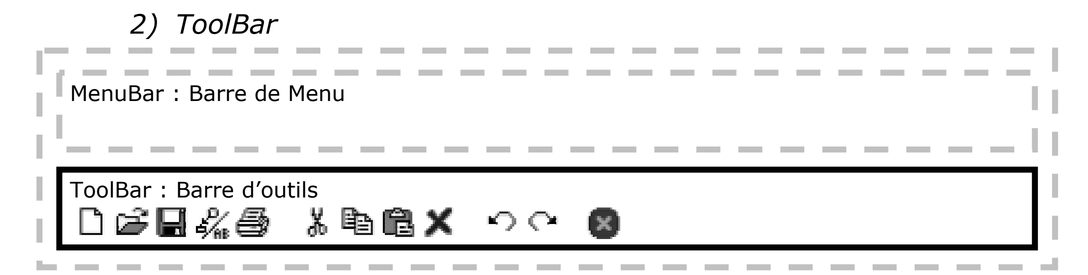


Figure 14 : schéma de la barre d'outils

Nouveau : Permet d'ouvrir une nouvelle fenêtre avec un réseau vide.

Ouvrir : Permet d'ouvrir un réseau de Petri précédemment enregistré.

Enregistrer : Enregistre le réseau courant dans un fichier.

Générer : Génère une image XML du réseau.

Imprimer : Permet d'imprimer le réseau courant sous sa forme graphique ou d'image XML.

Couper : Coupe la sélection courante.

Copier : Copie la sélection courante.

Coller : Colle le contenu de notre presse-papier en haut à gauche du réseau.

Supprimer : Supprime la sélection.

Annuler : Annule la dernière action effectuée sur le réseau.

Rétablir : Rétablit une action annulée auparavant.
Fermer : Ferme le réseau courant.

b) *Afficheur*

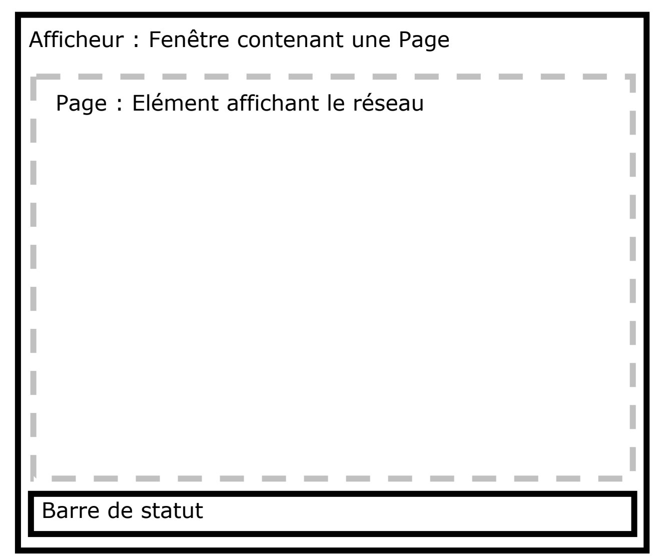


Figure 15 : schéma de l'afficheur

Cette fenêtre permet, comme son nom l'indique, d'afficher un réseau via une Page. C'est cette fenêtre qui gère les évènements claviers et souris ainsi que les menus contextuels. Pour des raisons de commodités, la Page est contenue dans une entité qui permet de faire défiler le réseau, dans le cas où celui-ci serait plus grand que l'Afficheur.

1) Page

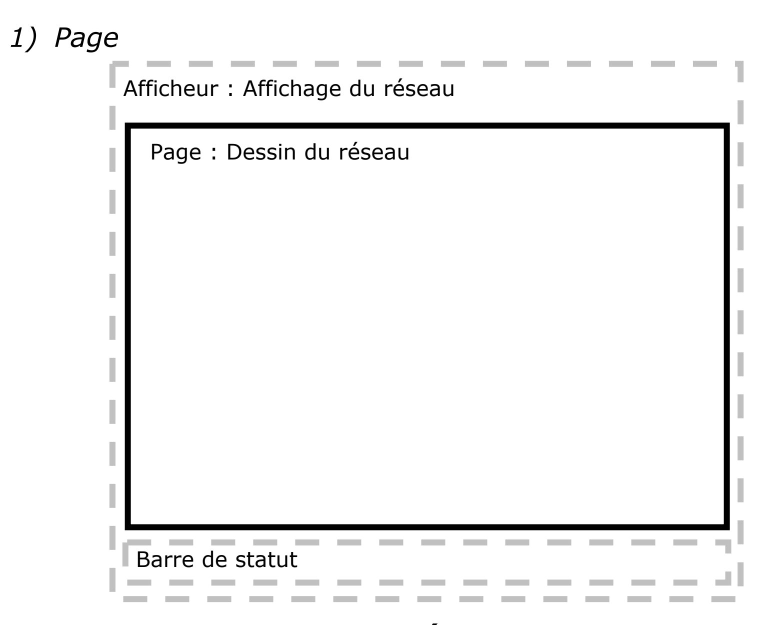


Figure 16 : schéma de la page

C'est la partie la plus importante d'un Afficheur, elle contient le réseau, crée les éléments de celui-ci, les listes gérant l'annulation/rétablissement d'une action. En fait elle gère quasiment tout ce qui est relatif au réseau lui-même sauf les propriétés des éléments. En résumé elle gère :

- La création des éléments graphiques du réseau.
- Le dessin du réseau.
- Les actions comme l'impression, le couper/copier/coller/supprimer, l'annulation ou le rétablissement d'une action, la sélection d'un objet.

c) *GraphicBar*

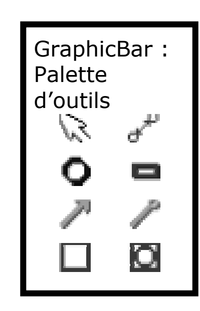


Figure 17 : schéma de la palette d'outils

Cette fenêtre est l'une des plus utiles pour utilisateur. En effet elle lui permet de créer les éléments graphiques du réseau. On peut donc créer ainsi une Place, une Transition, un Arc, un Arc inhibiteur, un Arc de Test, un Arc de fusion, un Sous-Réseau et aussi choisir l'outil de sélection.

d) *Information*

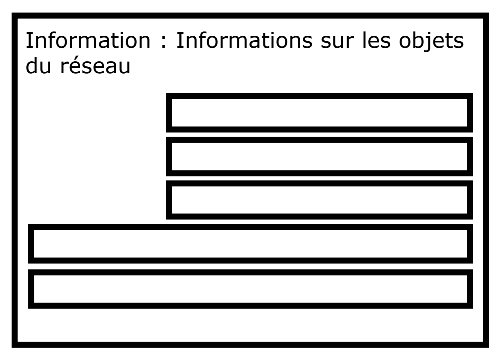


Figure 18 : schéma de la fenêtre d'informations des entités graphiques

Cette fenêtre a pour but d'afficher les informations des objets graphiques lorsque l'utilisateur passe le pointeur de la souris sur un de ceux-ci. Voici la liste des informations que cette fenêtre affiche :

- Le nom de l'objet, celui-ci remplace le titre de la fenêtre.
- Le type de l'objet pointé.
- L'identifiant de l'objet.
- Les coordonnées (abscisse, ordonnée) du lieu où se trouve l'élément.

e) *InfoClasses*

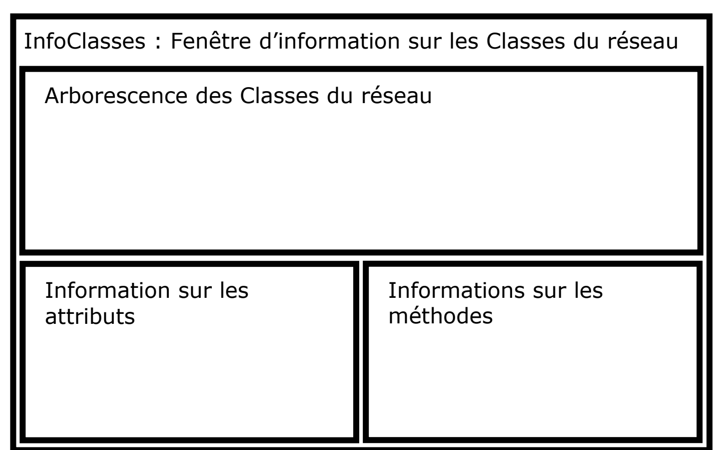


Figure 19 : schéma de la fenêtre d'informations des classes

Cette fenêtre est composée de trois parties, la première qui se trouve en haut est la principale, elle affiche la hiérarchie des Classes du réseau, lorsque l'utilisateur sélectionne une classe dans l'arbre, la liste de ses attributs s'affiche dans la liste en bas à gauche et la

liste des méthodes s'affiche en bas à droite. Lorsque l'utilisateur double-clique sur une des classes, sa fenêtre de Propriétés s'ouvre.

f) *Propriété*

Cette fenêtre permet à l'utilisateur de gérer les propriétés des éléments du réseau, elle lui permet de changer le nom de l'objet, de changer sa position et bien d'autres choses. Voici la liste des caractéristiques qui peuvent être changées via cette fenêtre :

- Transition : On peut changer la liste des Arcs entrants ou sortants, le Verrou qui lui est associée, sa fenêtre temporelle et surtout les Actions et Conditions.
- Place : Comme pour une Transition il est possible de changer la liste des Arcs entrants ou sortants, de modifier sa position, de stipuler si c'est une Place de sortie, d'entrée ou interne. On peut aussi changer les tuples de Classes que cette Place accepte et évidemment modifier le marquage.
- Arc : Pour cet élément, il est seulement possible de changer son étiquette en fonction de sa Place aval ou amont.
- Classe : Il est possible via cette fenêtre de changer la liste des attributs, des méthodes, de modifier l'héritage de la classe et choisir dans quelle librairie sera sauvée cette classe.

4. Réalisation

4.1. *Editeur de RdPO*

Comme il a été dit auparavant, l'éditeur a été développé en Java, il s'appuie donc sur une approche objet afin que cet éditeur soit modulaire et facile à faire évoluer. De plus le langage Java fournit toute une série de bibliothèques qui permettent de créer facilement des interfaces.

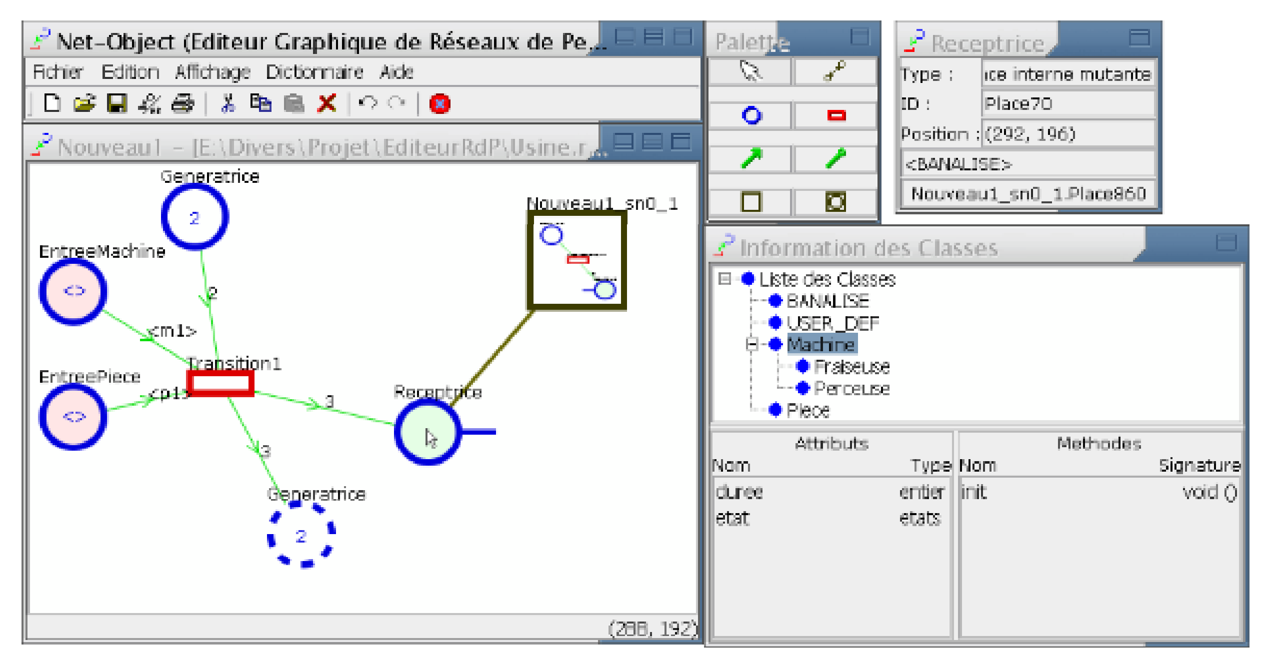


Figure 20 : vue générale de l'interface

4.1.1. Les entités d'un RdPO

Voici la description des éléments manipulés par l'interface, cette partie est très importante car elle conditionne le bon développement de l'interface de l'éditeur.

a) *Super*

Comme son nom l'indique, cette classe est la parente de toutes celles qui seront listées par la suite. Afin de pouvoir identifier les éléments d'un réseau cette classe a un compteur, permettant d'avoir l'unicité d'un identifiant dans un réseau.

b) *SuperLieu*

Cette classe abstraite définit tous les éléments graphiques du réseau, elle a des méthodes permettant de déplacer un élément, de le renommer etc. Les Noeuds, les Arcs héritent donc de cette classe, qui a pour but de faciliter la détection et la manipulation d'éléments graphiques dans le réseau.

c) *Lieu*

Comme la classe précédente, celle-ci est abstraite, elle définit l'emplacement d'un élément graphique du réseau. Pour ceci, deux points A et B sont utilisés, selon les cas, ils définissent un « vecteur » ou un rectangle, dans le premier cas, A est le point de départ du « vecteur » et B l'arrivée, dans le second cas, A représente le coin supérieur gauche du rectangle et B son opposé, c'est-à-dire le coin inférieur droit. A ce propos, c'est à l'élément graphique lui-même de s'assurer de la cohérence de ces informations.

Cette classe contient essentiellement des méthodes permettant de déplacer ce lieu de manière absolue, en le plaçant en un endroit défini, ou alors de manière relative, en lui donnant deux valeurs qui indiquent un déplacement sur les abscisses et les ordonnées.

En plus de ces méthodes, cette classe fournit des méthodes permettant de tester l'inclusion d'un point ou d'un autre Lieu, dans celui-ci.

d) *LieuArc*

Cette classe hérite de Lieu, les points A et B définissent donc respectivement le départ et l'arrivée de l'Arc. La particularité de cette classe est que le test d'inclusion d'un point dans ce LieuArc se fait avec une marge d'erreur. Sinon il serait trop difficile de sélectionner un Arc.

e) *LieuBoite*

Cette classe héritant de Lieu, les points A et B définissent un rectangle. Ce type de Lieu est utilisé pour les Noeuds (Places et Transitions) ainsi que pour les RdP.

f) *Arc*

Cette classe implémente l'élément graphique Arc du réseau, il a comme attributs deux Noeuds (un amont et l'autre aval), un LieuArc définissant sa position sur le réseau et une liste d'Etiquettes. Plusieurs méthodes sont fournies, une permettant de renvoyer le type de l'Arc selon que celui-ci soit un Arc Post (Transition → Place) ou un Arc Pre (Place → Transition), une autre permettant de connaître la pondération éventuelle de cet Arc, si il ne peut pas porter de Jetons banalisés cette méthode retournera -1, il faudra alors accéder à la liste des Etiquettes afin de savoir quels sont les tuples d'objet pouvant être portés par cet Arc.

g) *ArcTest*

La classe ArcTest hérite de Arc, elle pour seule particularité de renvoyer ArcTest comme type.

h) *Inhibiteur*

Cette classe par rapport à Arc (dont elle hérite) a la seule particularité de renvoyer Inhibiteur comme type.

i) *ArcFusion*

Ce type d'Arc n'a pas du tout le même sens que les trois types d'Arcs vus précédemment, en effet ceux-ci relient des réseaux à des sous-réseaux afin de fusionner deux places ou deux transitions.

Par contre, de manière analogue aux types d'Arcs précédents, ces Arcs de fusion ont un type, c'est 'rr' dans le cas où cet Arc fusionne deux Noeuds qui se trouvent dans deux sous-réseaux de même niveau (c'est-à-dire au même niveau hiérarchique), c'est 'pr' quand la fusion a lieu entre un Noeud d'un réseau et un Noeud de l'un de ses sous-réseaux direct, 'rp' quand la fusion a lieu entre un Noeud d'un sous-réseau et un Noeud de son réseau englobant (cf. Figure 21).

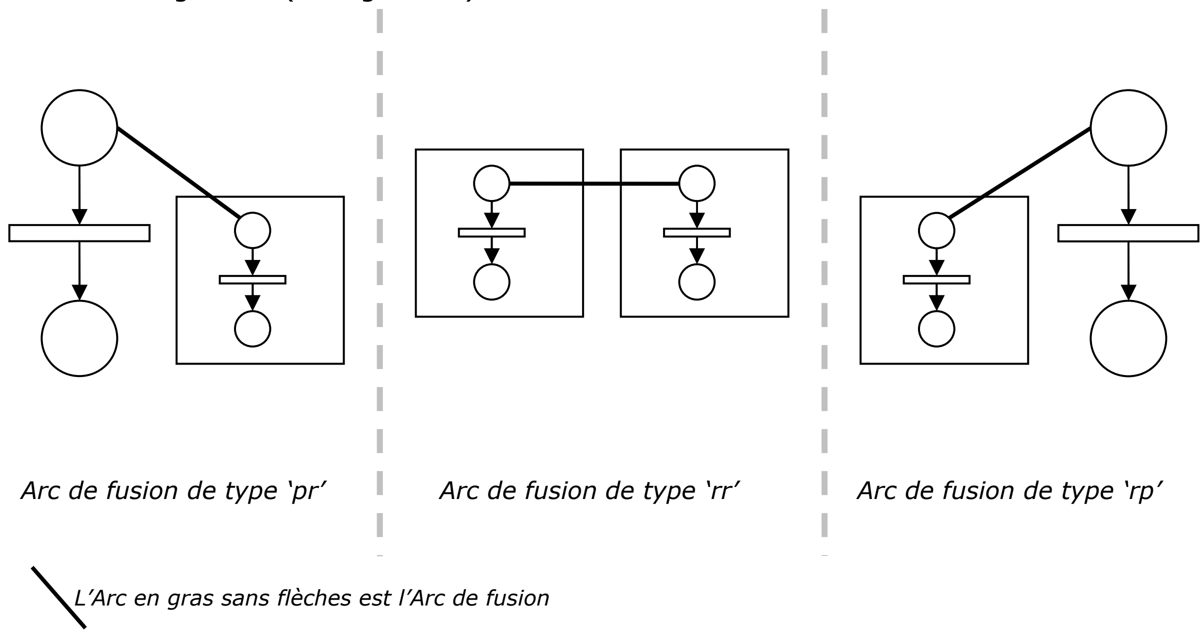


Figure 21 : les différents types d'arcs de fusion

j) *Methode*

Cette classe implémente les méthodes associées aux Classes d'un réseau elle a pour attributs un nom, une signature et le chemin du fichier C/C++ contenant le code du corps de la méthode (les méthodes étant développées en dehors de l'éditeur de RdPO).

k) *Classe*

Cette classe implémente les Classes du réseau de Petri, comme il l'est défini dans le modèle, elle a pour attributs un nom, une liste d'attributs, une liste de Méthodes, ainsi qu'une éventuelle Classe parente. Elle fournit aussi des méthodes qui permettent d'obtenir les attributs et les méthodes de ses ancêtres.

l) *Etiquette*

Cette classe a une utilité différente selon la classe (Arc ou Place) qui l'utilise. Ella a pour attributs une liste dont un élément sur deux (le premier) est le nom de la variable qui a pour type la Classe placée en deuxième position (cf. Figure 22). Or lorsque l'Etiquette est associée à une Place, le nom de la variable est ignoré, c'est pour cela que cette classe a deux méthodes pour le test d'égalité, une qui teste tous les éléments de la liste et l'autre qui ne teste que les Classes.

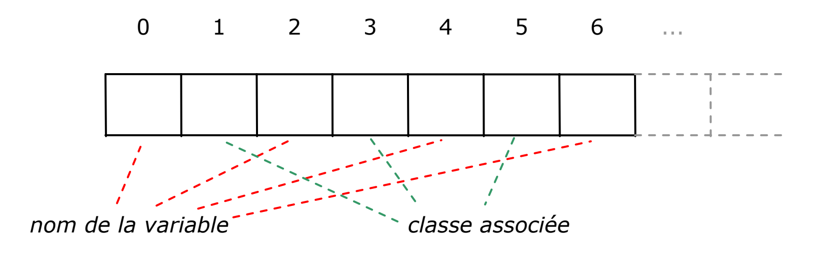


Figure 22 : structure d'une etiquette

m) *Noeud*

Noeud est la classe dont hérite Place et Transition, elle a pour attributs un nom, une liste d'Arcs entrants, une liste d'Arcs sortants ainsi qu'un LieuBoite. Un Noeud peut avoir plusieurs type : 'normal' si il a des Arcs entrants et sortants, 'isole' si il n'a ni Arc sortants ni entrants, 'puit' si il a seulement des Arcs entrants et 'source' si il a seulement des Arcs sortants.

n) *Objet*

Implémente l'instanciation d'une Classe d'un réseau, cette classe a pour attribut la Classe à laquelle cet Objet est associé (ceci afin de pouvoir mettre à jour cet objet si la Classe est modifiée), une liste correspondant aux valeurs des attributs de la Classe et le nom de cette instance.

o) *Jeton*

Cette classe représente les tuples d'Objet. C'est seulement une liste d'Objet.

p) *Place*

Cette spécialisation de Noeud implémente les Places du réseau. Elle pour attributs la liste des Arcs de fusion, la liste des Etiquettes qui définit les Jetons pouvant marquer cette Place, la liste de ses Places clones et un autre indiquant si il s'agit d'une Place d'entrée, de sortie ou interne.

Comme la classe Arc il existe une méthode permettant de savoir le nombre de Jetons banalisés pouvant se trouver dans cette Place, cette méthode retourne 0 si la Place peut contenir des Jetons banalisés, -1 si elle ne peut en contenir ou un nombre supérieur à 0 indiquant leur nombre. Pour connaître la liste des Classes acceptées par cette Place, il faut se référer à la liste d'Etiquettes.

q) *PlaceClone*

Cette spécialisation de Place implémente les Places clones du réseau. Elle a pour attribut la Place originelle. Les méthodes ne concernant pas les Arcs ont été redéfinies afin d'accéder aux méthodes correspondantes de la Place originelle. En ce qui concerne les Arcs, les listes sont différentes que ce soit un clone ou une Place originelle.

r) *Transition*

Cette classe est une autre spécialisation de Nœud elle implémente les Transitions du réseau. Les principaux attributs sont la liste des Arcs de Fusion, la liste des Conditions, celle des Actions, la fenêtre temporelle, le verrou, la priorité de cette Transition ainsi que booléen indiquant si l'action doit être exécutée en paralléle.

s) *RdP*

Cette classe abstraite dont RdPSimple et RdPComposite sont une spécialisation est l'implémentation de ce que représente un réseau. Elle a pour attributs un nom, un booléen indiquant si ce réseau est comprimé ou non (si le réseau est sous la forme de sous-réseau à l'écran), un booléen indiquant si ce réseau est générique, un LieuBoite déterminant la localisation du RdP sur le réseau quand il est sous forme de sous-réseau, le chemin de sauvegarde du réseau et enfin le chemin du fichier de librairie de Classes.

Cette classe fournit beaucoup de méthodes comme une permettant d'obtenir le RdPSimple d'un réseau, il en existe qui renvoie la liste des éléments du réseau selon leur type, une autre permet de savoir si un nom est utilisé par un élément d'un certain type, et d'autres permettant de connaître ce qui se trouve à un endroit du réseau, etc.

t) *RdPSimple*

Cette classe, spécialisation de RdP implémente les réseaux simples. Elle a pour attributs une liste d'Arc, de Noeuds, de Classes, de Verrous et un RdPSimple représentant son père éventuel dans la hiérarchie des sous-réseaux. Des méthodes existent mais c'est seulement des redéfinitions des méthodes de RdP.

u) *RdPComposite*

Cette classe qui est aussi une spécialisation de RdP représente les réseaux composés de sous-réseaux ou alors les sous-réseaux eux-mêmes. Elle a pour principaux attributs une liste de RdP, une liste d'Arc de fusions entre nœuds du RdPComposite et une liste d'Arcs de fusion dont l'une des extrémités n'est pas dans ce réseau composite. Une méthode existe renvoyant un RdPComposite selon son nom, les autres méthodes sont des redéfinitions des méthodes de RdP.

4.1.2. L'interface de l'éditeur

Cette partie est peut-être la plus sensible du projet, étant donné que c'est elle qui centralise toutes les autres et les coordonne afin d'avoir un fonctionnement correct et respectant les attentes de l'utilisateur. Heureusement, Java fournit un grand nombre de classes qui permettent de faire ce type d'interface relativement facilement.

a) *Editeur*

Cette fenêtre qui est une spécialisation de JFrame gère aussi un grand nombre d'évènements, comme ceux relatifs au clavier (raccourcis claviers), aux actions (les boutons), aux fenêtres ou aux menus. Il gère aussi la liste des Afficheurs, l'ouverture d'un réseau ou d'un sous-réseau, la création des verrous etc. Les menus (JMenu et JMenuItem) se trouvant sur la barre de menus (JMenuBar) permettent à l'utilisateur d'utiliser facilement ces fonctionnalités. La barre d'outils (JToolBar) affiche des boutons (JButton) qui correspondent aux fonctionnalités les plus utilisées (cf. Figure 23).



Figure 23 : Editeur

L'Editeur gère aussi la palette d'outils graphiques (cf. Figure 24), c'est une JDialog celle-ci est constituée de huit boutons (JButton) sur lesquels se trouvent des icônes (ImageIcon) afin de faciliter leur identification.

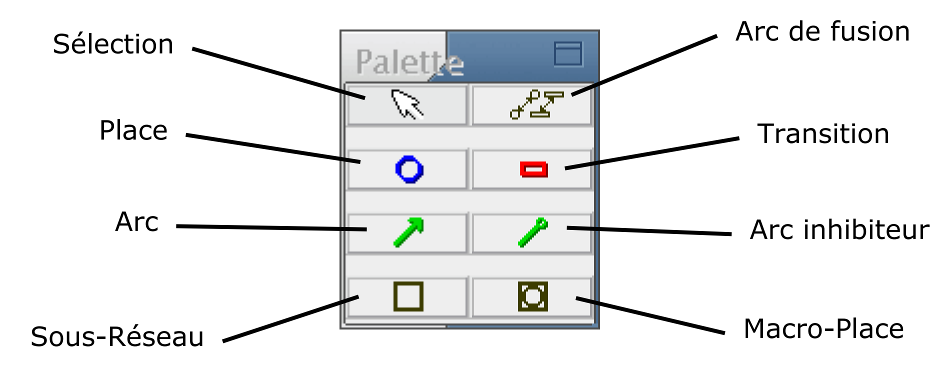


Figure 24 : palette d'outils

b) *Information*

Cette spécialisation de JDialog permet à l'utilisateur d'afficher des informations relatives à l'élément du réseau qui est pointé par la souris. Elle est constituée de trois JLabel qui affichent le type des informations qui sont affichées dans les trois premiers champs textuels (JTextField). Ces champs affichent le type de l'élément, son identifiant et sa position sur la Page. Les deux autres champs textuels affichent des informations pour certains types d'éléments, par exemple pour les Arcs le 4ème composant affiche leur Etiquette. Le 5ème champ affiche les informations relatives à la fusion pour une Place. Il est à noter que le nom de l'élément pointé devient le titre de cette fenêtre.

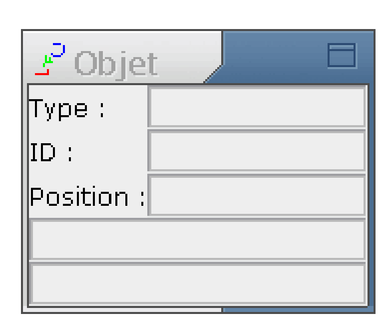


Figure 25 : information sur les entités

c) *InfoClasses*

Cette extension de JDialog affiche les informations concernant les Classes du réseau contenu dans l'Afficheur courant. Etant donné que les Classes sont organisées en une hiérarchie nous avons choisi d'utiliser un composant d'affichage d'arbres (le JTree). Pour afficher les informations relatives aux attributs et aux méthodes des Classes nous avons opté pour quatre listes (deux pour les attributs et deux pour les méthodes) : la première affiche le nom des attributs, la deuxième affiche le type des attributs, la troisième affiche le nom des méthodes et la dernière liste affiche leur signature.

Outre ces attributs, il existe des méthodes qui mettent à jour et construisent l'arbre à partir d'un Vecteur de Classes et d'autres qui mettent à jour l'affichage des attributs et des méthodes lorsqu'une Classe est sélectionnée.

Lorsque l'utilisateur double-clique sur le nom d'une Classe de l'arborescence la fenêtre de Propriete de cette Classe s'affiche.

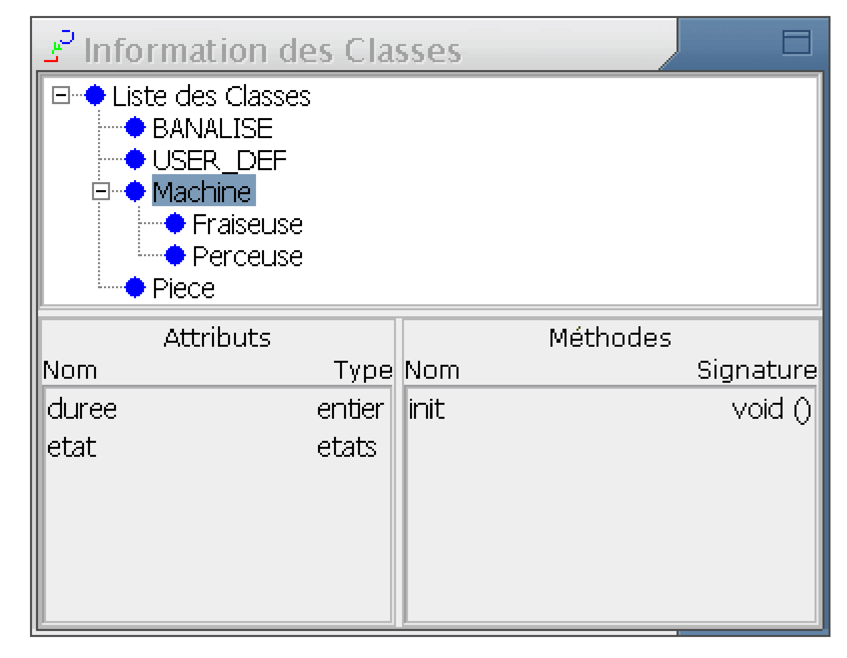


Figure 26 : information des Classes

d) *Afficheur*

Cette fenêtre a pour principale fonction d'afficher une Page et de la gérer. Elle est donc composée d'une Page, d'un composant qui permet de faire défiler cette Page quand le réseau qu'elle affiche est plus grand que l'Afficheur lui-même, de deux composants (deux JLabel) qui constituent la barre de statut ainsi que de trois composants (une JWindow, un JTextArea et un JPanel) servant à afficher une petite aide contextuelle.

En plus d'afficher une Page, l'Afficheur gère le menu contextuel, les raccourcis claviers, les évènements souris qui permettent de créer, de déplacer les éléments constitutif du réseau, etc.

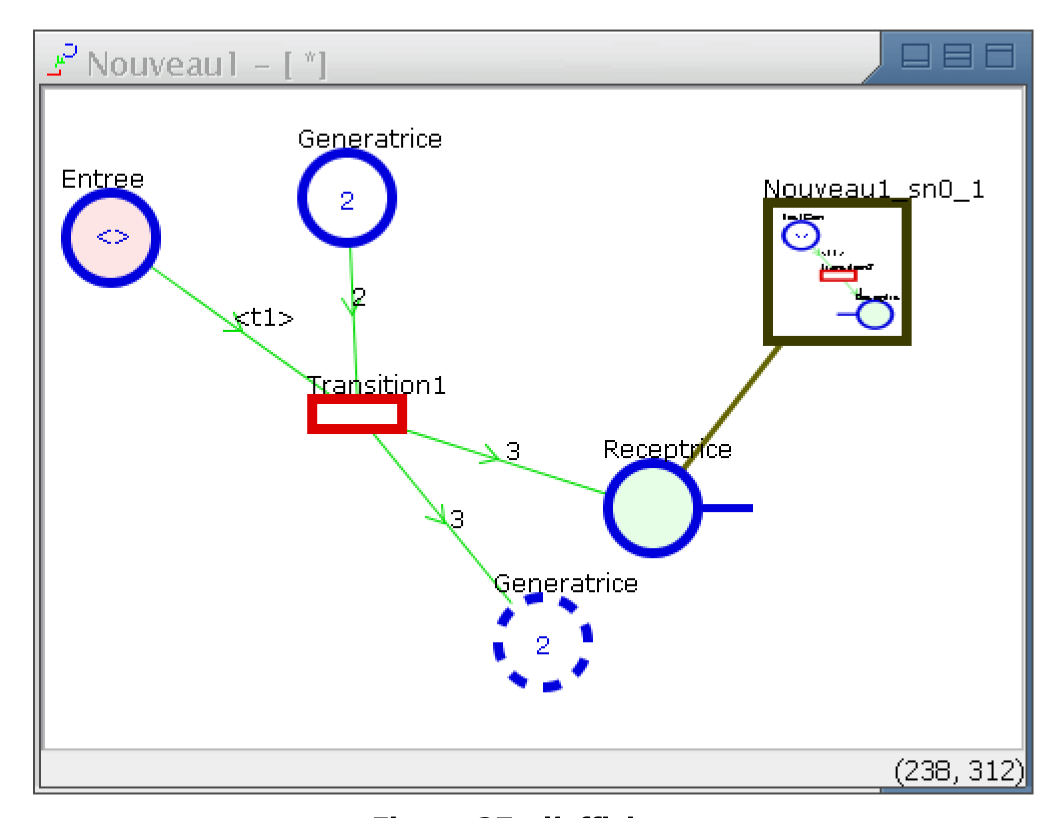


Figure 27 : l'afficheur

e) *Page*

La Page est la classe qui contient et gère un réseau. A part le RdP elle a pour attributs un Vecteur qui contient la sélection courante d'éléments du réseau, quatre piles (Stack) qui servent à gérer la fonctionnalité d'annulation/rétablissement d'une action faite sur le réseau. En plus de contenir le réseau, de le gérer et d'en créer les éléments, Page dessine aussi les Arcs, les Places, les Transitions etc. Elle permet d'imprimer la forme graphique du réseau, elle gère les fonctionnalités de copier/couper/coller/supprimer.

f) *ModifListe et SelectListe*

Ces deux classes sont beaucoup utilisées dans l'édition des propriétés des éléments du réseau. SelectListe (Figure 28) permet de choisir un ou plusieurs composants dans une liste.

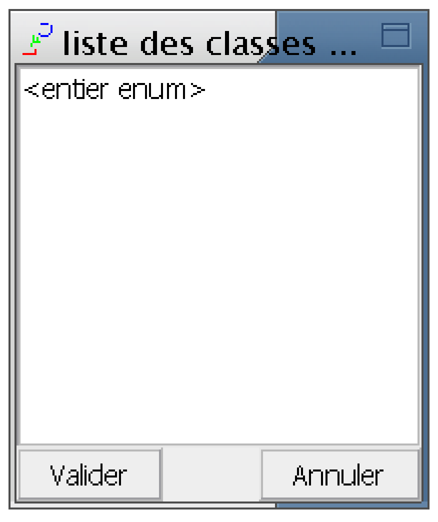


Figure 28 : SelectListe

ModifListe (Figure 29) permet de modifier le contenu d'une liste, la colonne de gauche affiche les valeurs disponibles et la colonne de droite affiche la liste que l'on édite cliquer sur '==>' pour ajouter l'élément sélectionné à gauche dans la liste de droite, cliquer sur '<==' pour supprimer un élément de la liste de droite.

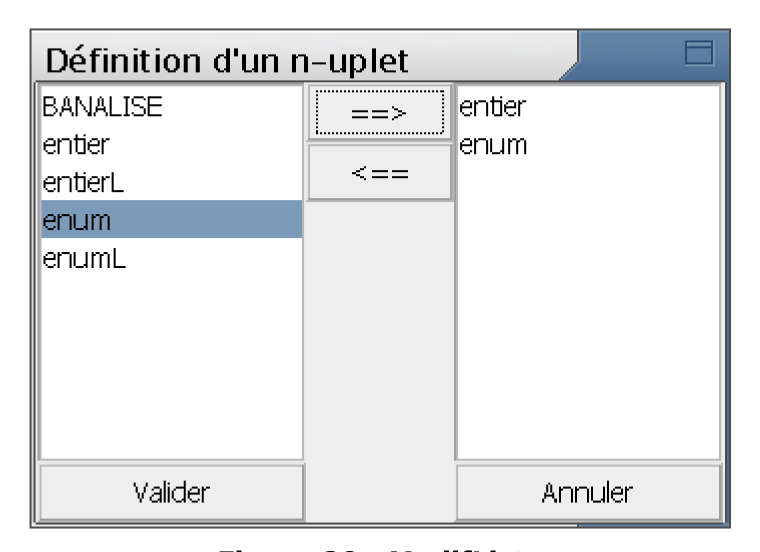


Figure 29 : ModifListe

g) *Propriete*

La classe Propriete est l'une des classes les plus importantes de l'éditeur, car c'est elle qui gère toutes les propriétés des composants d'un réseau. Elle vérifie aussi que les données entrées par l'utilisateur sont valides, surtout en ce qui concerne les Transitions. Elle une extension de JDialog, afin de faciliter sa description nous allons détailler les propriétés de chaque composant.

1) Classes

La fenêtre des propriétés des Classes (Figure 30) est composée de listes (JList) des méthodes et attributs, de boutons (JButton) afin d'éditer/supprimer/créer un attribut ou une méthode. Bien évidemment, il est possible de nommer la Classe, de choisir sa parente et de choisir la librairie dans laquelle sera sauvée cette Classe.

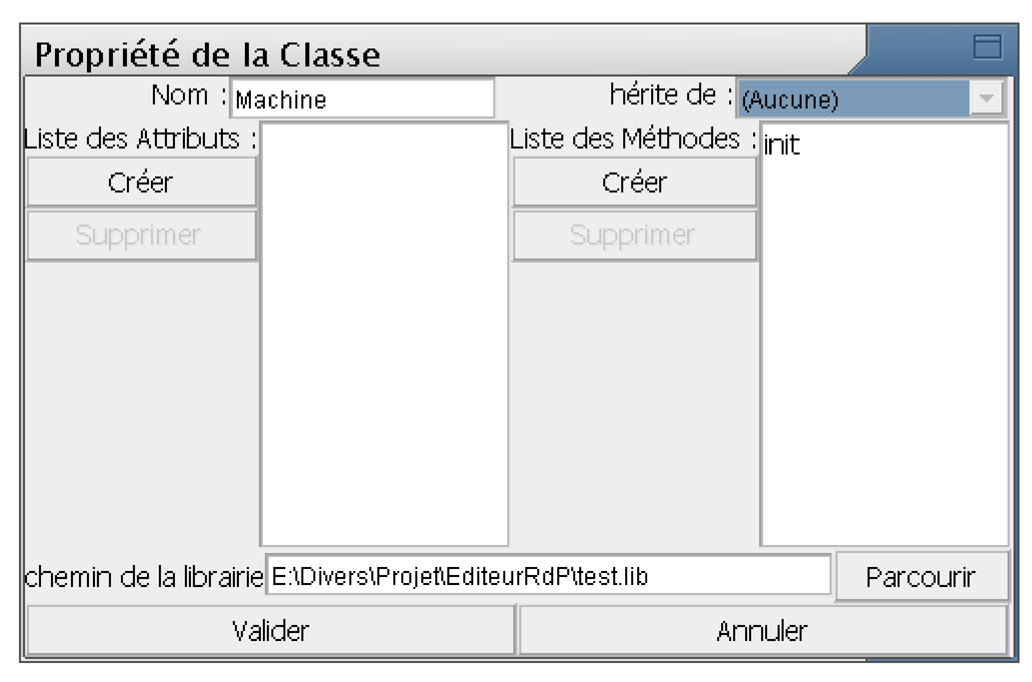


Figure 30 : propriétés de la Classe

Lorsque l'utilisateur choisit de créer ou d'éditer une méthode, cette fenêtre (Figure 31) s'affiche. Elle est composée de trois boutons (JButton) et de trois champs (JTextField). Ces derniers permettent de rentrer le nom, la signature et le chemin.

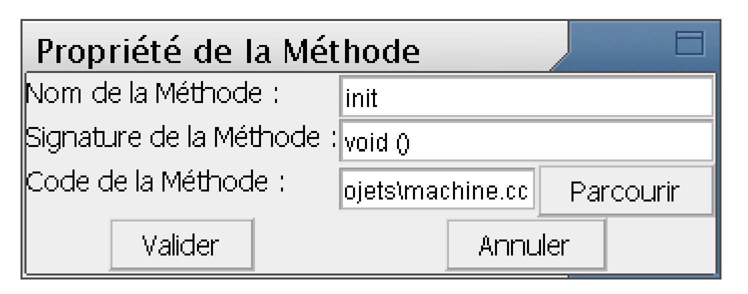


Figure 31 : propriétés de la méthode

Lorsque l'utilisateur choisit de créer ou d'éditer un attribut, cette fenêtre (Figure 32) s'affiche. Il y a un champ (JTextField) pour entrer le nom de cet attribut, une case (JCheckBox) qui permet de choisir si il s'agit d'une liste, deux boutons radios qui permettent de choisir entre un attribut de type « simple » ou un attribut de type énuméré. Si 'Type Simple' est sélectionné l'utilisateur peut choisir le type parmi ceux qui sont dans la boîte déroulante (entier, reel, string, booleen ou un type énuméré déjà défini) (JComboBox). Si 'Type Enuméré' est sélectionné, l'utilisateur pourra définir un nouveau type énuméré en entrant des valeurs dans la liste à l'aide des boutons fléchés et du champ à droite, le champ de gauche permet de nommer cette énumération.

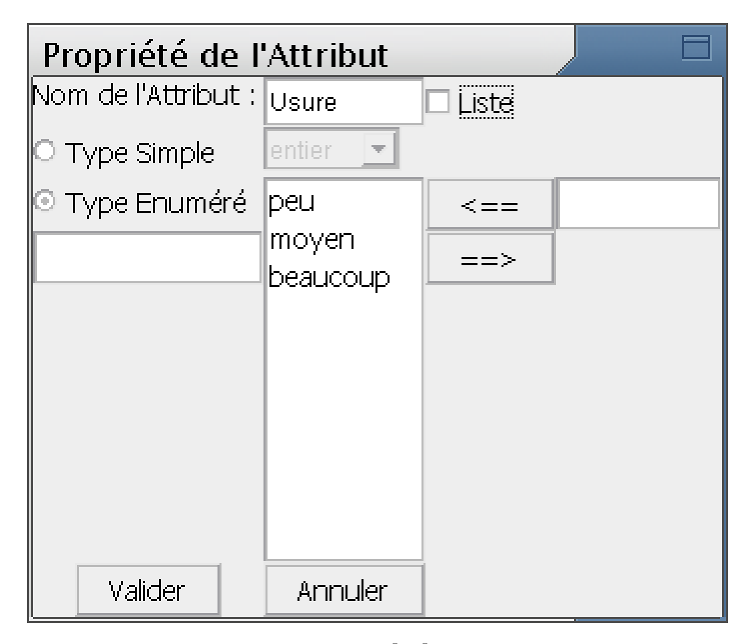


Figure 32 : propriétés de l'attribut

2) Places

Les fenêtres de Propriété pour les Places (Figure 33), les Arcs et les Transitions sont similaires. Celle des Places a plusieurs champs (JTextField) qui permettent de changer le nom, la position, ou d'afficher son identifiant il est possible de changer le type de la Place en fonction du nombre d'Arcs. Les boutons (JButton) permettent de modifier la liste des Arcs entrants/sortants à l'aide d'une ModifListe (**fonctionnalités à utiliser avec prudence, car aucune vérification n'est faite, sur la cohérence des changements**), les tuples de Classe acceptés par la Place (avec une ModifListe) et le marquage.

Propriété de la Place
Nom : Place16
Identifiant : Place16
Position X : 80
Position Y : 71
Type : ISOLE
Modifier la Liste des Arcs Entrants
Modifier la Liste des Arcs Sortants
Modifier les classes de la Place
Modifier le marquage de la Place
Valider
Annuler

Figure 33 : propriétés de la Place

En appuyant sur le bouton 'Modifier les classes de la Place' l'utilisateur pourra créer ou éditer un tuple de Classe qui désignera les Classes qui sont acceptées par les Places.

En appuyant sur le bouton 'Modifier le marquage de la Place' la fenêtre ci-après s'affichera (Figure 34). Dans la colonne (une JList) de gauche s'affichent les tuples de Classes acceptés par la Place. Lorsque l'utilisateur sélectionne un de ces tuples, les Classes associées à ce tuple s'affichent dans la liste du milieu. La sélection de l'une de ces Classes permettra de donner un nom à l'instance en entrant un nom dans le champ du bas puis en cliquant sur 'OK'. En sélectionnant le nom de l'attribut dans la liste de droite et en cliquant sur 'Editeur/créer' il est possible de changer la valeur d'un attribut.

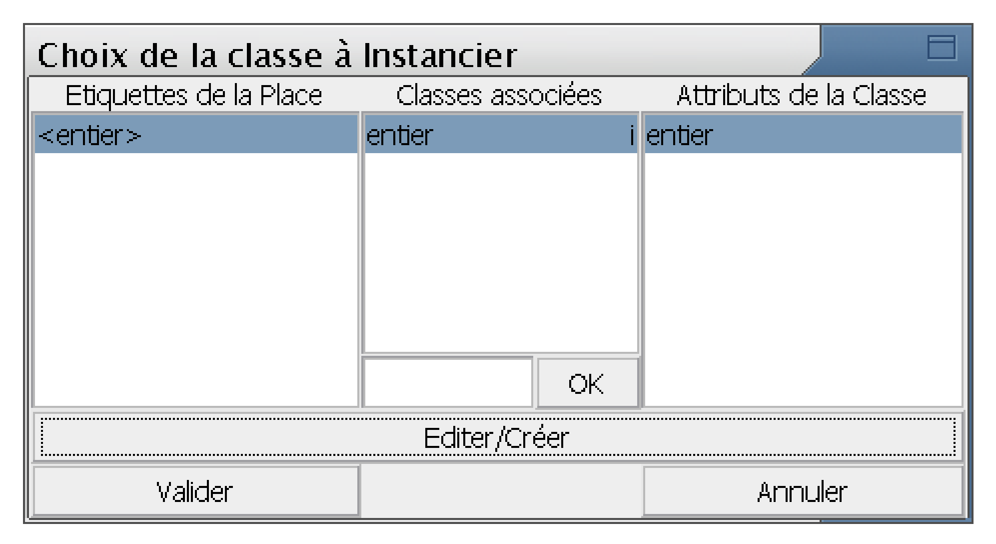


Figure 34 : instanciation d'une Classe

3) Arcs

La fenêtre de Propriété pour un Arc (Figure 35) est plus réduite que les autres, elle affiche l'identifiant et la position de celui-ci et ne comporte qu'un seul bouton (à part 'Valider' et 'Annuler') permettant de modifier la liste des variables.

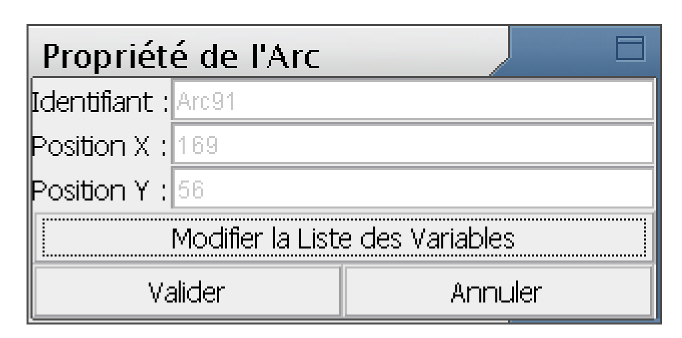


Figure 35 : propriétés de l'Arc

Cette fenêtre (Figure 36) permet d'éditer les variables portées par les Arcs. En haut se trouve la liste des variables déjà portées par l'Arc. En cliquant sur 'Créer' ou 'Editer' (si une variable est sélectionnée) la liste des tuples des Classes acceptées par la Place aval ou amont s'affiche dans la liste en bas à gauche. Comme pour la fenêtre de marquage, si l'utilisateur sélectionne l'un des tuples, sa composition s'affichera dans la liste de droite et il pourra en éditer le nom de la variable en utilisant le champ en bas à droite et le bouton 'OK'. Cliquer sur le bouton ajouter, ajoutera l'étiquette à la liste du haut.

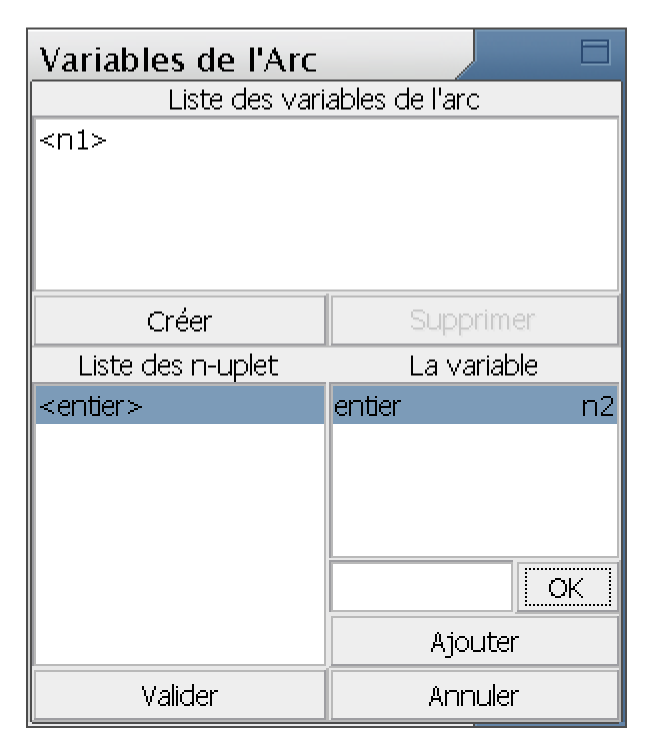


Figure 36 : variables sur les Arcs

4) Transitions

La fenêtre de Propriete des Transitions (Figure 37) est celle qui offre le plus d'options, c'est celle qui demande le plus de vérification. Elle affiche le nom de la Transition, son identifiant, sa position. Grâce à une boîte déroulante il est possible de choisir un verrou. La priorité de la Transition peut être changée via le JSpinner. Comme pour la Place, il est possible de modifier la liste des Arcs entrants/sortants. La fenêtre temporelle peut être éditée avec le bouton 'Modifier la fenêtre de Temps'.



Figure 37 : propriétés de la Transition

En appuyant sur le bouton 'Modifier les Prédicats', la fenêtre de la Figure 38 s'affichera. Son utilisation est relativement simple. En haut se trouve la liste des prédicats déjà entrés pour cette Transition. En dessous, il est possible d'entrer une expression booléenne en éditant les deux champs et en choisissant un opérateur, les champs ne peuvent contenir que des expressions du type `nom_variable.attribut`, `nom_variable.méthode` ou

`constante`, évidemment les expressions contenues dans les champs doivent être compatibles entre elles et avec l'opérateur choisi.

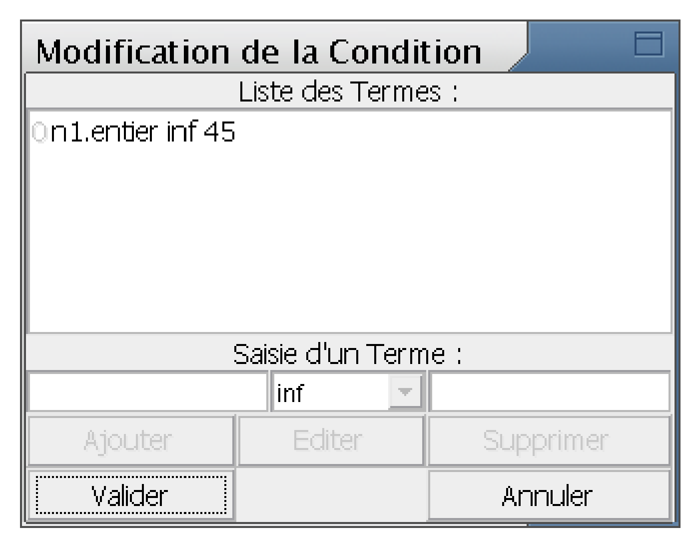
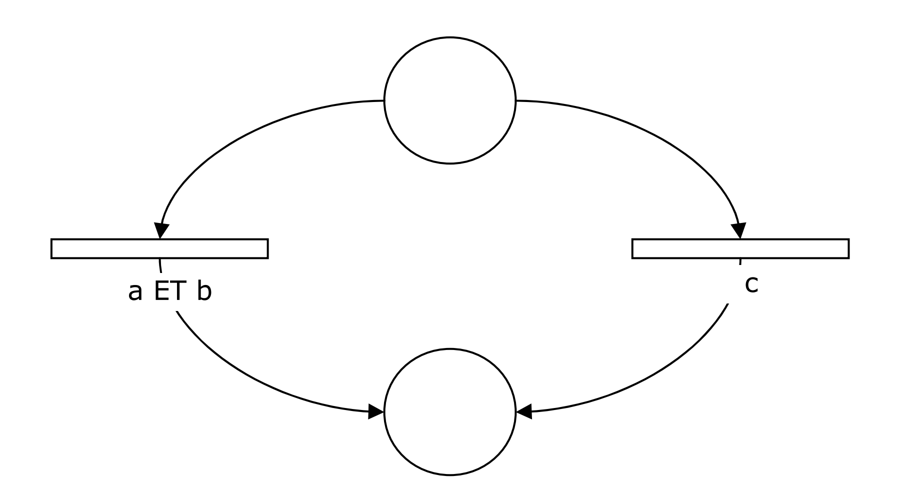


Figure 38 : conditions d'une Transition

Au final tous ces termes sont liés entre eux par l'opérateur booléen ET, si l'utilisateur veut exprimer un OU, il devra le faire structurellement (cf. Figure 39) :



L'absence de la place aval donnerait à la structure le sens d'un OU exclusif.

Figure 39 : expression structurelle du OU

Le bouton 'Modifier les Actions' permet d'accéder à la fenêtre ci-après (cf. Figure 40). De manière similaire à la précédente, en haut se trouve la liste des actions déjà présentes dans cette Transition. Par contre la saisie d'une action est un peu plus compliquée, en effet il faut d'abord choisir entre l'appel simple à une méthode et la définition d'une expression plus complexe. Les champs remplis par l'utilisateur sont analysés et vérifiés afin que les règles soient respectées (cf. Annexe).



Figure 40 : actions d’une Transition

5. Références

[David89] R. David, H. Alla. *Du grafcet aux réseaux de Petri*. Traité des Nouvelles, HERMES, 1989.
[Valette00] R. Valette. *Les Réseaux de Petri*. L.A.A.S. C.N.R.S. Toulouse, Septembre 2000.
[Passama02] R. Passama. *Rapport de Stage de DEA Informatique*. L.I.R.M.M. Montpellier, Juillet 2002.
[Raclot02] F. Raclot. Rapport de Stage de DESS TNI. L.I.R.M.M. Montpellier, Juillet 2002.
[RR02181] F. Raclot, D. Andreu, T. Libourel, R. Passama. *Manuel d'utilisation de l'éditeur E-NetObject*. RR L.I.R.M.M. n° 02181.
[RR02182] R. Passama, D. Andreu, F. Raclot, T. Libourel. *J-NetObject : un noyau d'exécution de Réseaux de Petri à Objets*. RR L.I.R.M.M. n° 02182.

ANNEXE

1. Règles concernant les conditions et les actions d'une Transition

Voici les règles à respecter pour une action ou une condition d'une Transition. Celles-ci sont vérifiées par l'éditeur (la plupart sont évidentes, mais il est bon de les rappeler) :

- Un terme est de la forme :
  - 1er `constante`
  - 2e `nom_variable.attribut`
  - 3e `nom_variable.methode(terme, terme,...)`
- une expression est de la forme :
  - 1er `terme`
  - 2e `expression opérateur expression`
- les expressions ainsi que les opérateurs doivent être compatibles.
- les paramètres passés aux méthodes doivent être conformes à la signature de celles-ci.

Quelques exemples :

Conditions :

x.a < 3
- 1° terme : x.a (`nom_variable.attribut`)
- opérateur : <
- 2° terme : 3 (`constante`)

y.b > z.c
- 1° terme : y.b (`nom_variable.attribut`)
- opérateur : >
- 2° terme : z.c (`nom_variable.attribut`)

y.c == z.code()
- 1° terme : y.c (`nom_variable.attribut`)
- opérateur : == (égalité)
- 2°terme : z.code() (`nom_variable.methode()`)

Actions :

x.a = y.b
- 1° terme : x.a (`nom_variable.attribut`)
- opérateur : = (allocation)
- 2° terme : y.b (`nom_variable.attribut`)

x.b = y.tirerchiffre()
- 1° terme : x.b (`nom_variable.attribut`)
- opérateur : = (allocation)
- 2°terme : y.tirerchiffre() (`nom_variable.methode()`)

z.inc()
- terme : z.inc() (`nom_variable.methode()`)

z.calcule(3, x.c)
- terme : z.calcule(3, x.c) (`nom_variable.methode(constante, nom_variable.attribut)`)